175 PTAS
mi computer
60
CURSO PRACTICO DEL ORDENADOR PERSONAL,
EL MICRO Y EL MINIORDENADOR
commodore
plus/4
Editorial Delta, S.A.

MI COMPUTER CURSO PRACTICO
DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen V - Fascículo 60

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones, Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Paseo de Gracia, 88, 5.°, 08008 Barcelona
Tels. (93) 215 10 32 / (93) 215 10 50 - Télex 97848 EDLTE

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-7598-007-4 (tomo 5)
84-85822-82-X (obra completa)
Depósito Legal: B. 52/1984

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 068503
Impreso en España - Printed in Spain - Febrero 1985

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 19 425 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 429 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S.A. (Paseo de Gracia, 88, 5.°, 08008 Barcelona), o también con talón bancario remitido a la misma dirección.

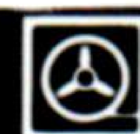
c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

Para cualquier aclaración, telefonear al (93) 215 75 21.

No se efectúan envíos contra reembolso.



Movimientos exactos

En esta oportunidad abordaremos diversos aspectos de la programación de brazos-robot

Un signo de los tiempos

Un brazo-robot que se hubiera programado para hacer firmas exhibiría movimientos tanto de camino continuo como de punto por punto. Pintar cada palabra habría de ser un movimiento continuo que le hubiera sido enseñado por un operador; al final de cada palabra, sin embargo, el brazo debería moverse directamente de un punto de la firma hasta el bote de pintura, y éste es un movimiento de punto por punto que el brazo puede calcular a partir de su posición en curso y la ubicación conocida del bote de pintura

Steve Cross

Ya hemos visto cómo se pueden construir brazos-robot que se asemejen a una extremidad humana: poseen "esqueleto" para proporcionar una estructura y "músculos" para proporcionar energía motriz. Pero el brazo aún necesita "inteligencia" para poder llevar a cabo tareas.

La idea de un brazo inteligente a primera vista parecería absurda. Sin embargo, la forma de inteligencia que estamos considerando aquí no corresponde a aquella de alto nivel que poseen los seres humanos, sino a algo considerablemente menos complejo. Tomemos a modo de ejemplo una acción humana sencilla. Usted se encuentra sentado junto a una mesa que está vacía, a excepción de un pequeño objeto que se halla colocado a la izquierda. Su tarea consiste en trasladar este objeto desde la parte izquierda hasta la parte derecha de la mesa. Aquí intervienen dos formas de inteligencia. La primera implica la percepción tanto de la mesa como del objeto y la decisión de desplazar éste de uno a otro lado. Esto presupone un "pensamiento consciente", y está relacionado con conceptos tales como "intención" y "comportamiento orientado hacia un objetivo". La inteligencia que necesitamos considerar es muy elemental: la que se precisa para mover el brazo y la mano correctamente *después* de haber decidido la tarea a cumplir: desplazar la mano hasta la posición correcta y asegurarse de que coja el objeto y lo suelte en el momento preciso.

Entrenamiento humano

Esto parece a la vez sencillo y obvio; pero si tiene alguna duda en el sentido de que este acto sea realmente inteligente, tan sólo observe a un niño pequeño tratando de seguir la misma secuencia. El pequeño a menudo no conseguirá asir el objeto, lo desplazará hasta una posición inadecuada y es probable que parezca bastante inseguro sobre qué es lo que se requiere. El niño está intentando adquirir la inteligencia necesaria para mover sus brazos y manos en un mundo tridimensional que le es extraño. Después de aprendido esto, tales movimientos se realizarán de forma automática, sin exigir ningún pensamiento consciente, y entonces dejará de pensar en ellos como si exigieran inteligencia.

El brazo-robot se encuentra en la misma situación que el niño que empieza a andar: posee el equipo para realizar tareas, pero debe "aprender" a realizarlas de forma automática.

El método más sencillo consiste en entrenar el brazo para que efectúe labores específicas con sólo guiarlo a lo largo de una secuencia de movimientos y decirle que "la recuerde". Este procedimiento se utiliza en una gran cantidad de robots industriales. Un operador toma literalmente de la mano al robot y lo conduce a través de los pasos que debe dar. Esto tiene la enorme ventaja de que la persona que lo "instruye" no necesita saber nada sobre cómo funciona en realidad el brazo-robot: todo lo que ha de saber es la secuencia de acciones que debe seguir el brazo. Por su parte, el robot no necesita "saber" lo que está haciendo: simplemente tiene que "recordar" las acciones que debe llevar a cabo.

Geometría de dos junturas
Al moverse desde un punto a otro, el brazo-robot de dos junturas debe girar alrededor de su pivote (ángulo R) y debe cambiar los ángulos del hombro (H) y del codo (C). Si las coordenadas cartesianas del punto actual y del punto de destino fueran (X1, Y1, Z1) y (X2, Y2, Z2), los cambios se calcularían del siguiente modo:

```
A1=SQR(X1^2+Y1^2+Z1^2)
A2=SQR(X2^2+Y2^2+Z2^2)

Pivote:
R1=ARCTAN(Y1/X1)
R2=ARCTAN(Y2/X2)
Cambio=(R2−R1)

Hombro:
H1=ARCCOS(Z1/A1)+
ARCCOS((A1^2+U^2−
L^2)/(2*A1*U))

H2=ARCCOS(Z2/A2)+
ARCCOS((A2^2+U^2−
L^2)/(2*A2*U))
Cambio=(H2−H1)

Codo:
C1=ARCCOS((U^2+L^2−
A1^2)/(2*U*L))
C2=ARCCOS((U^2+L^2)−
A2^2)/(2*U*L))
Cambio=(C1−C2)
```

donde U y L son las longitudes del brazo superior e inferior respectivamente

EJE Z
C1
H1
H2
C2
U
L
(X2,Y2,Z2)
EJE Y
(X1,Y1,Z1)
R1
R2
EJE X

Kevin Jones

Métodos de entrenamiento

Hay dos clases de "entrenamiento" que se utilizan con los brazos-robot, el de punto por punto y el de camino continuo. En el entrenamiento de *punto por punto*, el operador mueve el brazo hasta una cierta posición y luego pulsa un botón para señalarle al robot que debe "recordar" esa posición. El brazo es movido hasta la siguiente posición y se vuelve a pulsar el botón. Esta secuencia prosigue hasta haber almacenado en la memoria del autómata cibernético una secuencia completa de acciones. Una vez concluida la sesión de entrenamiento, se pone al robot en modalidad de "reproducción" *(playback)* y éste entonces se mueve de un punto al otro exactamente en la manera en que se le "enseñó". En el entrenamiento de *camino continuo*, el operador conduce al robot a través de la secuencia completa y éste recuerda todas y cada una de las posiciones de la secuencia. Al reproducirla, el robot sigue la secuencia de la misma forma que antes.

Retomando el ejemplo de una persona que esté sentada junto a una mesa y trasladando un objeto de uno a otro extremo de ésta, podemos utilizar el método de punto por punto para "entrenar" a un robot de modo que reproduzca esta acción. Este procedimiento se emplea con frecuencia con robots que deban llevar a cabo tareas de "asir y colocar": trasladar un objeto de un lugar a otro. Por el contrario, un robot que se utilice para atomizar pintura necesitará que se le enseñe mediante el método de camino continuo para asegurar que cubra con pintura el objeto en su totalidad, tal como lo haría una persona.

Ahora vamos a considerar cómo "recuerda" el robot la secuencia de movimientos que debe seguir. La respuesta para ello es que el robot utiliza sensores internos para grabar la posición de cada una de sus junturas durante la modalidad de entrenamiento. Esto a menudo se realiza tomando la salida de los codificadores de eje y grabando los movimientos efectuados, ya sea directamente en la memoria o bien, para un almacenamiento más permanente, en cinta o en disco. Cuando se selecciona la modalidad de reproducción, el robot puede, entonces, recuperar los datos correspondientes y convertirlos en movimiento de junturas, lo que constituye una tarea relativamente compleja.

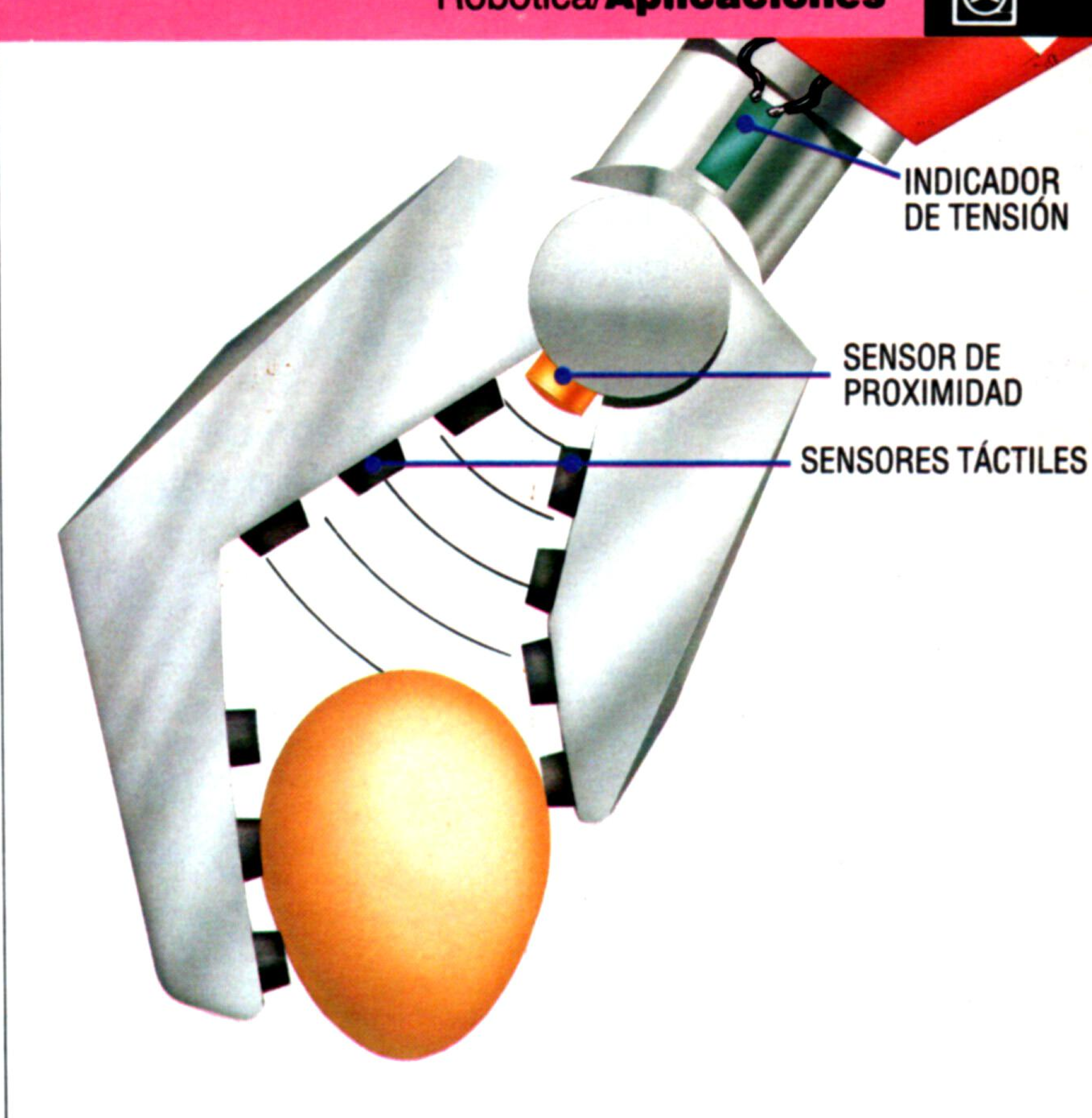
Resulta sorprendente el hecho de que al robot le resulte más fácil "recordar" el movimiento de camino continuo: sólo tiene que seguir la ruta exacta que se le ha enseñado. No obstante, con frecuencia es necesario almacenar una cantidad de datos muy grande; a menudo se necesitan varios centenares de posiciones para definir un camino continuo, en vez de las pocas posiciones implicadas en el movimiento de punto por punto. La segunda dificultad deriva del hecho de que, si el brazo ha de seguir la secuencia uniforme y exactamente, todas sus junturas deben activarse simultáneamente. Un robot que atomice pintura tendrá que hacer un barrido del brazo a lo largo de los tres ejes, al tiempo que maniobra sus tres junturas de muñeca para posicionar el atomizador de la forma correcta. Esto significa que el ordenador que controle al robot debe trabajar muy rápido con el fin de manipular cada juntura por turno sin ninguna demora apreciable; por otra parte, el robot puede utilizar hasta seis procesadores separados, dirigiendo cada uno de ellos el movimiento de una juntura, para obtener un movimiento perfectamente simultáneo.

Movimiento calculado

Un robot de punto por punto tiene ante sí una tarea más ardua, porque si bien sabe hacia dónde moverse, no se le ha "dicho" cómo llegar hasta allí. Podría, simplemente, mover cada juntura hasta colocarla en la posición requerida, pero esto supondría una pérdida tanto de tiempo como de energía. Sería mucho mejor que el robot calculara una ruta directa para su mano desde un punto a otro; entonces podría realizar el movimiento requerido en una barrida, tal como lo hace una persona. Pero, nuevamente, los cálculos necesarios para hacer esto son complejos, dado que se deben utilizar coordenadas cartesianas para mover la mano en una línea recta entre dos puntos definidos, mientras las posiciones propias del brazo se definen según un sistema de coordenadas totalmente diferente. De modo que el robot debe ser capaz de resolver algunos in-

trincados problemas geométricos con el fin de trabajar eficazmente. Y, en el caso de robots industriales que deben trasladar a gran distancia objetos que pesan varios cientos de kilos, puede ser considerable el ahorro en tiempo y energía que supone la elección de la mejor ruta.

Otro problema que se le plantea al robot de punto por punto es la dinámica del brazo. Si usted mueve un brazo para coger un objeto, comprobará que se acelera lentamente al alejarse de su posición original hasta alcanzar una velocidad máxima, luego desacelera hasta llegar a una suave detención en su posición final. Las ventajas que ofrece un brazo-robot que hace esto son considerables. Muchos de tales brazos se mueven a una velocidad constante, acelerando casi inmediatamente hasta una velocidad máxima y deteniéndose en seco al final de la secuencia de movimientos. Esto supone un esfuerzo del brazo y exige más energía que otro que acelere y desacelere suavemente. Asimismo, significa que el brazo quizá no se mueva tan rápidamente como lo haría en el caso contrario y, si al final de la secuencia se requiere que el robot coja un objeto delicado, incluso un imperceptible desplazamiento del objeto podría provocar que el brazo golpeara contra él con una fuerza considerable. De modo que es preciso calcular una velocidad óptima para el robot además de un camino ideal a seguir.



Steve Cross

Suavemente
La prueba de coger un huevo es decisiva para juzgar los sensores del brazo-robot y los mecanismos de control de realimentación. El sensor de proximidad de la uña debe comprobar que el huevo esté lo suficientemente cerca como para cogerlo, luego los dedos pueden empezar a acercarse hasta que los sensores táctiles indiquen el contacto con el huevo. La salida de los sensores táctiles se debe entonces comparar con la del sensor de proximidad a medida que los dedos se cierran y el brazo comienza a levantarse. Una disminución repentina de la proximidad indica que el huevo se está escurriendo, de modo que los dedos deben apretar hasta llegar a un límite de fuerza de agarre preestablecido o hasta que una súbita disminución de la fuerza de agarre indique que la cáscara de huevo se está deformando antes de romperse

Coreografía de robots

Incluso después de haber programado un brazo para que siga una serie precisa de movimientos, en el momento de la reproducción puede resultar que estos movimientos no respondan exactamente a los requerimientos. Esto puede deberse a error humano, a que la naturaleza de la tarea haya cambiado ligeramente o bien a que, si el robot está trabajando en conjunción con otros brazos, éstos sigan su propio camino y choquen entre sí. (Evitar este último problema es lo que se conoce como *coreografía de robots*.) De manera que se requiere un método para editar la secuencia. Esto se puede conseguir almacenando los movimientos como una lista encadenada, en la cual cada posición del brazo está almacenada junto con la dirección en donde se encuentra la posición siguiente. En la sesión de entrenamiento inicial, esta dirección será la de la siguiente posición de la lista. Si se necesitara editar la secuencia, se podría desplazar el brazo hasta la posición en la cual fuera necesario realizar las correcciones, detenerlo e insertar una nueva secuencia.

Otro método común para hacer que los brazos se muevan inteligentemente consiste en utilizar una serie de instrucciones programadas almacenadas en el ordenador. Típicamente, cada robot posee su propio método de programación y emplea un "lenguaje" de programación diferente para controlar el movimiento, pero en general se requiere un lenguaje que le permita al programador utilizar instrucciones tipo LOGO para especificar el movimiento en tres dimensiones, con instrucciones adicionales para movimiento de muñeca y efector final, tales como "asir" o "depositar".

El problema es similar al de entrenar un robot de punto por punto y es necesario tener en cuenta muchos factores. Por ejemplo, si se ha de mover un brazo-robot 10 unidades hacia adelante, el método más obvio sería alterar la juntura del hombro de modo que el brazo pudiera llegar más adelante. Sin embargo, esto haría que el brazo se moviera hacia arriba en un arco y, por lo tanto, esto se habría de corregir mediante un movimiento descendente en la juntura del codo. A partir de este ejemplo se puede ver que la instrucción para un solo movimiento simple se debe traducir en dos juegos distintos de instrucciones, trabajando en dos junturas separadas.

Se puede plantear otro problema cuando se requiere que el robot recoja un objeto. Por muy bien posicionado que esté el brazo, es difícil asegurar que esté exactamente en el lugar correcto para coger el objeto, en especial si éste tiene forma asimétrica. Por consiguiente, se necesita una mano "inteligente"; ésta debe percibir la presencia o la ausencia del objeto, la distancia del mismo respecto a la mano y la fuerza ejercida por ésta cuando intenta recogerlo. Estos problemas se pueden abordar equipando a la mano con una gama de sensores de proximidad, táctiles y de fuerza, que proporcionen la realimentación que capacite al ordenador controlador para efectuar cualquiera modificación necesaria.

Si se consideran todos estos problemas, podemos ver que es posible construir un brazo-robot que muestre un nivel de "inteligencia" relativamente elevado. Sin embargo, hasta el momento no se ha diseñado ningún brazo que, pongamos por caso, lance una pelota de cricket con precisión. Ello se debe a que la inteligencia del brazo por sí sola no es suficiente. El robot debe conocer, asimismo, la posición del bateador, la fuerza y la dirección del viento y un sinnúmero de otras condiciones variables. Luego deberá ser capaz de calcular las ecuaciones, complejas, que implica enviar un proyectil por el aire. Para dichas tareas se requiere mucho más que un mero brazo inteligente.



Espacio para maniobrar

En la programación de utilidades complejas es inevitable recurrir al uso del lenguaje máquina

Con el fin de operar nuestro anterior programa de búsqueda de variables lo mezclamos con el programa en el cual se deseaba efectuar la búsqueda. Con este método, la única información relativa al sistema operativo que teníamos que proporcionar era la dirección de comienzo del programa en BASIC; el fin del programa en el cual se realizaba la búsqueda se hallaba al encontrar el número de línea más bajo del programa de utilidad.

La utilidad que estamos creando es un programa para sustituir variables. Éste es un programa muy útil para tener en el archivo. Si a lo largo de todo un programa usted hubiera empleado un nombre de variable, descubriendo después que el mismo no es legal, imagínese cuánto tiempo le ahorraría una utilidad de esta clase. Igualmente, puede ser que se haya escrito un programa y que ahora vaya a ser utilizado por otra persona, a quien tal vez no le resulte fácil descifrar los nombres de las variables. Aquí le explicamos la teoría necesaria para el código máquina y en el próximo capítulo publicaremos los listados.

Espacio de memoria

En este ejercicio necesitamos colocar el programa de utilidad en una sección de memoria distinta de aquella en la que está trabajando el programa. Asimismo, debemos hallar un método diferente de localizar el final del programa en BASIC, y un medio para acomodar en el ordenador dos programas en este lenguaje al mismo tiempo.

Los tres ordenadores que estamos considerando (BBC Micro, Commodore 64 y Sinclair Spectrum) utilizan un conjunto de apuntadores para indicar al sistema operativo y al intérprete de BASIC dónde encontrar los programas en BASIC, las variables, etc. (véase p. 536). Lamentablemente, los detalles son distintos en las tres máquinas.

En el BBC Micro hay cuatro apuntadores importantes: PAGE y TOP, que retienen la dirección de comienzo y de final del programa en BASIC; LOMEM, que retiene la dirección de comienzo de las variables de BASIC, e HIMEM, que retiene la dirección del final del área para BASIC. Estos cuatro apuntadores se almacenan como variables de BASIC incorporadas, y podemos leer o alterar sus valores mediante sencillas sentencias en este lenguaje. Si tenemos en la memoria un programa en BASIC y deseamos agregar otro, cambiamos PAGE a un valor superior a TOP (utilizando la instrucción OLD para reinicializar TOP y LOMEM) y entonces podemos agregar el nuevo programa sin afectar al programa original. Pasamos de un programa a otro dándole nuevos valores tanto a PAGE como a HIMEM y utilizando OLD.

Una vez que tenemos en ejecución el programa de utilidad, los valores de los apuntadores se refieren a este programa; para permitir que la utilidad encuentre el comienzo y el final del programa sobre el cual está trabajando, hemos de copiar los valores originales en un área de la memoria que no se altere cuando cambiemos los programas. Otro procedimiento para hallar el final de un programa es utilizar el indicador final que coloca el intérprete de BASIC. Éste es simplemente un byte que retiene un valor de 128 o más, inmediatamente después del carácter de retorno de carro del final de la última línea del programa. Este byte, y el que le siga, se interpretarán como los bytes HI y LO del número de línea siguiente. Dado que el byte HI de este número es 128 o más, éste dará un número de línea mayor o igual a 32768 (256×128). Como el máximo número de línea válido es 32767, podemos estar seguros de haber hallado el indicador de final del programa y no tan sólo otro número de línea.

El Commodore 64 utiliza siete apuntadores, almacenados en la memoria de página cero, para indicar diversas partes del área para programas en BASIC. TXTTAB, en las direcciones 43 y 44, apunta al comienzo del programa en BASIC; VARTAB, ARYTAB, STREND, FRETOP y FRESPC, de la dirección 45 a la 54, apuntan a diversas secciones de la tabla de variables, y MEMSIZ, en las direcciones 55 y 56, apunta al final del área de BASIC. Se pueden cambiar estos apuntadores con el fin de crear un área separada en el cual ejecutar un programa en BASIC mediante el empleo de la instrucción POKE. Sin embargo, se recomienda un breve programa en lenguaje máquina, porque es más directo y reduce considerablemente las probabilidades de "colgar" el ordenador por algún error de digitación.

En el Commodore 64 el final de un programa en BASIC se indica mediante dos bytes que contienen ceros inmediatamente después del byte cero que indica el final de la última línea. Siguiendo la cadena de apuntadores del principio de cada línea del programa hasta hallar un apuntador con cero, obtendremos el final del programa.

Para el Spectrum

La creación de esta utilidad es algo más complicada en el Spectrum. En vez de un área para el programa en BASIC, hay un único bloque de memoria que no sólo incluye el programa y las variables en BASIC, sino también todas las áreas de trabajo que utilizan el sistema operativo y el intérprete de BASIC. Con este trazado de la memoria es difícil, si no imposible, tener dos programas en BASIC en el área operativa principal, de modo que haremos una copia de

nuestro programa por encima de RAMTOP y trabajaremos allí. Ello aún nos deja sin resolver el problema de recuperar el programa e instalarlo en el área principal de programas después de haberlo alterado, y necesitaremos un programa en código máquina para que haga esto por nosotros.

El manual del Spectrum proporciona muchísima información sobre la forma en que se almacena un programa en BASIC y para qué se utilizan las diversas áreas de la memoria. Sin embargo, debido a la gran cantidad de secciones diferentes del área operativa, y a la forma en que se pueden desplazar estas áreas, resulta difícil escribir programas de utilidad sin emplear subrutinas en código máquina tomadas de la ROM. Si desea hacer en el Spectrum algo de programación de utilidades seria, una valiosa obra de referencia es *The complete Spectrum ROM disassembly*, de Ian Logan y Frank O'Hara. En dicha obra se explica cómo funcionan todas las rutinas de ROM.

Experimentando con BASIC

Usted puede intentar alterar el contenido de un programa durante su ejecución, pero debe guardar el programa primero, porque el resultado más común es una caída del sistema. Utilice el programa Monitor (véase p. 598), que permite inspeccionar y modificar el contenido de la memoria. Ésta se puede inspeccionar y alterar a sí misma siguiendo sus instrucciones. Inserte algunas líneas REM extras al principio del programa y pruebe en ellas estas sugerencias:

- Hallar el comienzo del área para textos en BASIC (véase p. 538) e inspeccionar el programa Monitor en la memoria hasta identificar líneas de programa.
- Cambiar los valores de los bytes después de un distintivo REM, salir luego del programa y listar la línea modificada.
- Tratar de colocar un valor mayor que 127 en una línea REM; salir y listar: quizá se sorprenda.
- Alterar los bytes de número de línea de una línea; esto produce resultados imprevisibles, en especial si el nuevo número no sigue la secuencia de sus vecinos.
- Se pueden alterar los bytes de longitud de línea, pero debe insertarse un nuevo indicador de final de línea en el byte indicado.
- En el Commodore 64 se pueden cambiar los bytes de dirección de enlace: intente sustituir la dirección de enlace de una línea por la dirección de la siguiente, y después liste el programa.
- Si es más ambicioso, consulte su manual y explore el área de almacenamiento de las variables. Ésta suele empezar en el mapa de memoria allí donde termina el área para textos de BASIC. Hay hasta seis tipos distintos de variables, cada uno de ellos con su propio formato de almacenamiento: variables numéricas, matrices numéricas, variables de enteros, matrices de enteros, variables en serie y matrices en serie. Los formatos de las variables en serie y de enteros son los más simples, siendo esencialmente representaciones directas de los datos y el nombre de la variable; los datos de las matrices numéricas son los más complicados.
- Puede tratar de modificar los valores de los distintivos en las líneas del programa: esto modificará la palabra que representa una instrucción.

FE DE ERRATAS
En la página 598, en *Complementos al BASIC* del BBC y el Commodore:

- En la línea 1150 hay dos asignaciones sucesivas a C$(3); cambie la segunda asignación por
  C$(4)="Q"
- En la línea 6600, cambie Z=1 por
  Z=2
- La línea 200 de los complementos para el BBC debe ser
  200 CLS
  como en la versión para el Spectrum

Almacenamiento en BASIC

En el área para programas en BASIC, la mayoría de los micros siguen el mismo formato de almacenamiento. Cada línea de programa comienza con los datos de la línea: el número en forma de dos bytes y alguna información sobre la longitud. El texto del programa se almacena más o menos sin alteraciones, si bien las palabras clave se reemplazan por códigos de un byte denominados *distintivos*

500 REM **EJEMPLO**

VERSIÓN BBC

	HI	LO														
13	1	244	17	32	REM	*	*	E	J	E	M	P	L	O	*	*

INDIC. COMIENZO LÍNEA — NÚMERO DE LÍNEA — LONG. DE LA LÍNEA — DISTINTIVO — TEXTO DE PROGRAMA

Este formato 13 (código ASCII para [RETURN]) es un indicador de comienzo de línea; lo más común es que esté colocado al final de la línea. La longitud de ésta sólo ocupa un único byte. Se ha almacenado el espacio que está directamente después del número de línea

VERSIÓN COMMODORE 64

LO	HI	LO	HI													
18	8	244	1	REM	*	*	E	J	E	M	P	L	O	*	*	0

DIRECCIÓN DE ENLACE — NÚMERO DE LÍNEA — DISTINTIVO — TEXTO DE PROGRAMA — INDIC. FINAL LÍNEA

Los bytes de dirección de enlace contienen la dirección en dos bytes del primer byte de la siguiente línea del programa. El área para texto comienza en la dirección 2049, y esta línea tiene 17 bytes de longitud, de modo que la dirección de comienzo de la línea siguiente es 2066

VERSIÓN SPECTRUM

HI	LO	LO	HI													
1	244	13	0	REM	*	*	E	J	E	M	P	L	O	*	*	13

NÚMERO DE LÍNEA — LONG. DE LA LÍNEA — DISTINTIVO — TEXTO DE PROGRAMA — INDIC. FINAL LÍNEA

La longitud de la línea ocupa dos bytes, ¡de modo que cada línea del programa podría tener 65 535 caracteres de largo! Aquí la longitud de línea es 13: los bytes de la línea incluyendo el byte de final de línea, pero sin contar los bytes de información sobre línea

Liz Dixon

De tortugas y diablos

En este capítulo de nuestro curso de LOGO nos corresponde analizar las excepcionales facilidades para manipulación de sprites que posee el LOGO Atari

Las facilidades del LOGO Atari para la manipulación de sprites son, en realidad, notables. El usuario puede disponer de una amplia gama de colores entre los cuales escoger, además de contar con una novedosa facilidad para detectar eventos.

El LOGO Atari tiene cuatro sprites, que se numeran del 0 al 3, con el 0 como la tortuga "por defecto". El manual Atari habla de éstos como tortugas en vez de como sprites, de modo que nosotros también los llamaremos tortugas.

TELL 1 convierte a la tortuga 1 en la tortuga *en curso*; en otras palabras, la tortuga 1 obedecerá cualquier instrucción que se le dé. Pruebe con:

```
TELL 1
FD 40
RT 90
TELL 2
BK 40
RT 90
TELL 3
RT 90
```

Es posible, no obstante, tener más de una tortuga en curso. Pruebe con:

```
TELL [1 2 3]
FD 50
```

Ahora estas tres tortugas obedecerán las instrucciones que se les ha dado.

Las cuatro tortugas tienen la forma de tortuga rotatoria, la forma 0, hasta que se defina otra. Utilizando el editor se pueden definir hasta 15 formas distintas, y asignárselas después a las tortugas. Estas formas definidas por el usuario no rotan cuando la tortuga gira.

Digitando EDSH 1 el editor quedará preparado para editar la forma 1. Usted se puede mover por la pantalla utilizando las teclas para el cursor. Pulsando la barra espaciadora se llenará una caja vacía o se vaciará una llena. Después de haber diseñado una forma, se la define pulsando <ESC>. La instrucción SETSH 1 les conferirá a las tortugas 1, 2 y 3 (las tortugas en curso) la nueva forma.

ASK permite dirigir una instrucción a una tortuga determinada sin cambiar las en curso. Pruebe con:

```
ASK 1 [FD 20]
```

y verá que sólo se mueve la tortuga 1. Ahora digite FD 20 y se moverán las tres, porque las en curso aún siguen siendo los números 1, 2 y 3.

A las tortugas además de un encabezamiento y una posición se les puede dar una velocidad. SETSP 30 le confiere a la tortuga en curso una velocidad de 30 en su dirección actual. Las tortugas mantendrán sus velocidades hasta que éstas sean cambiadas. Se pueden crear procedimientos, o crear dibujos en la pantalla, sin que por ello se alteren las velocidades. Para detener a las tortugas, se les da una velocidad de 0. Entrando al editor también se detendrán, puesto que la entrada del editor siempre destruye la visualización de gráficos, ya que comparten la misma área de memoria.

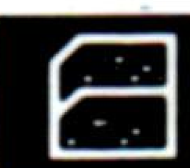
En el Atari existen 128 tonalidades diferentes de color a elegir. Se puede establecer el color de fondo, el color del lápiz y el color de la tortuga. Por ejemplo, SETBG 92 establecerá el fondo en verde. SETPC 0 23 establecerá el color del lápiz en naranja. El 23 es el código para naranja y el 0 es el número de lápiz. En LOGO Atari, la tortuga puede elegir entres tres lápices para escribir, si bien en este capítulo sólo utilizaremos el lápiz 0 (el lápiz por defecto). SETC 7 pondrá en blanco a la tortuga en curso. En el manual de Atari hay una tabla de colores y sus correspondientes códigos.

Diablos

El aspecto más original del LOGO Atari es su utilización de *diablos*. El LOGO puede detectar 21 *colisiones* y *eventos especiales*. La mayoría de éstos son colisiones entre tortugas o entre tortugas y líneas. Un diablo le dice al LOGO lo que ha de hacer cuando se produce una de estas colisiones. Por ejemplo, la colisión número 0 es cuando la tortuga número 0 cruza una línea dibujada con el lápiz número 0. Para preparar un diablo utilizamos la instrucción WHEN (cuando). Pruebe con:

```
CS
TELL 0
PD
FD 50
PU
RT 90
FD 100
RT 90
FD 20
RT 90
```





con la tortuga de cara a él. Ahora prepare un diablo **WHEN** con la instrucción siguiente:

```
WHEN 0 [BK 50]
```

Inmediatamente no sucede nada, pero ahora el diablo **WHEN** está presente dentro del ordenador manteniéndose alerta ante un posible evento 0. Ahora pruebe **SETSP 30**. La tortuga parte hacia la línea, pero cuando llega a ella se activa el diablo **WHEN** y la tortuga es repelida. Ésta continúa yendo a una velocidad de 30, pero cada vez que llega a la línea el diablo **WHEN** la rechaza.

Este diablo **WHEN** permanecerá en operación hasta que sea eliminado digitando **WHEN 0 [ ]**. Todos los diablos desaparecerán al digitar **CS**, si se produce un mensaje de error o al emplear el editor.

Es una incomodidad tener que recordar todos los códigos para las distintas colisiones, de modo que hay dos primitivas para servirle de ayuda: **OVER <númerodetortuga> <númerodelápiz>** produce el número para la colisión entre esa tortuga y una línea dibujada con ese lápiz, y **TOUCHING <númerodetortuga 1> <númerodetortuga 2>** produce el número del diablo para una colisión entre esas dos tortugas.

A la caza de tortugas

He aquí un conjunto de procedimientos para encerrar una tortuga dentro de una caja. Cada vez que la tortuga toca el borde de la caja (**WHEN OVER 0 0**) un diablo llama al procedimiento **GIRAR**. Éste hace que la tortuga retroceda 10 unidades y luego efectúe un giro al azar. (**RANDOM**, junto con un número, N, produce un número al azar entre 0 y N−1 inclusive.)

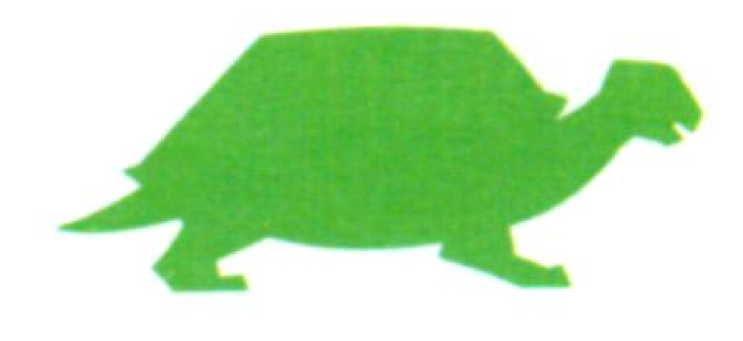
```
TO ATRAPAR
  DIBUJAR.TRAMPA
  HOME
  WHEN OVER 0 0 [GIRAR]
  SETSP 50
END

TO DIBUJAR.TRAMPA
  CS
  PU
  SETPOS [-50 -50]
  PD
  CUADRADO
  PU
END

TO CUADRADO
  REPEAT 4 [FD 100 RT 90]
END

TO GIRAR
  BK 10
  RT RANDOM 45
END
```

Los diablos también se pueden utilizar para vigilar la palanca de mando. De los 21 eventos especiales que hemos mencionado, el evento 3 se produce cuando se pulsa el botón de la palanca, y el 15 cuando cambia la posición de la misma. La instrucción **JOY 1** genera un número entre −1 y 7 que corresponde a la posición de la palanca (en la puerta 2). Defina **JOYH** de este modo:



```
TO JOYH
  IF(JOY 1)<0[STOP]
  ASK 0[SETH 45*JOY 1]
END
```

y ponga entonces en movimiento a la tortuga con **SETSP 50**; por último, prepare un diablo **WHEN**:

```
WHEN 15 [JOYH]
```

Ahora se pueden utilizar las palancas de mando para controlar el encabezamiento de la tortuga 0.

Se puede generar más de una instrucción **WHEN** al mismo tiempo, pero no estarán activas simultáneamente. Mientras un diablo está ocupado (es decir, cuando se produce su evento), los otros están inactivos. Por tanto, puede que se produzcan algunas colisiones sin que se las detecte.

La forma de tratar este problema consiste en hacer que cada diablo establezca las velocidades en 0 y ejecutar un procedimiento continuo que vigile que esto suceda. Adaptemos el programa anterior para aplicar esta técnica:

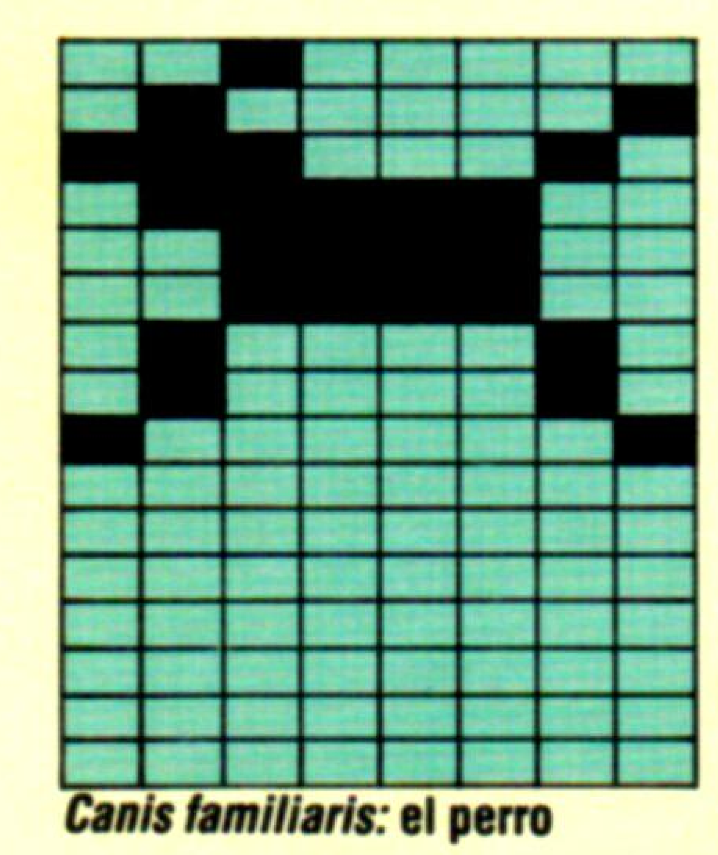
```
TO ATRAPAR
  DIBUJAR.TRAMPA
  HOME
  WHEN OVER 0 0 [SETSP 0]
  SETSP 50
  VIGILAR
END

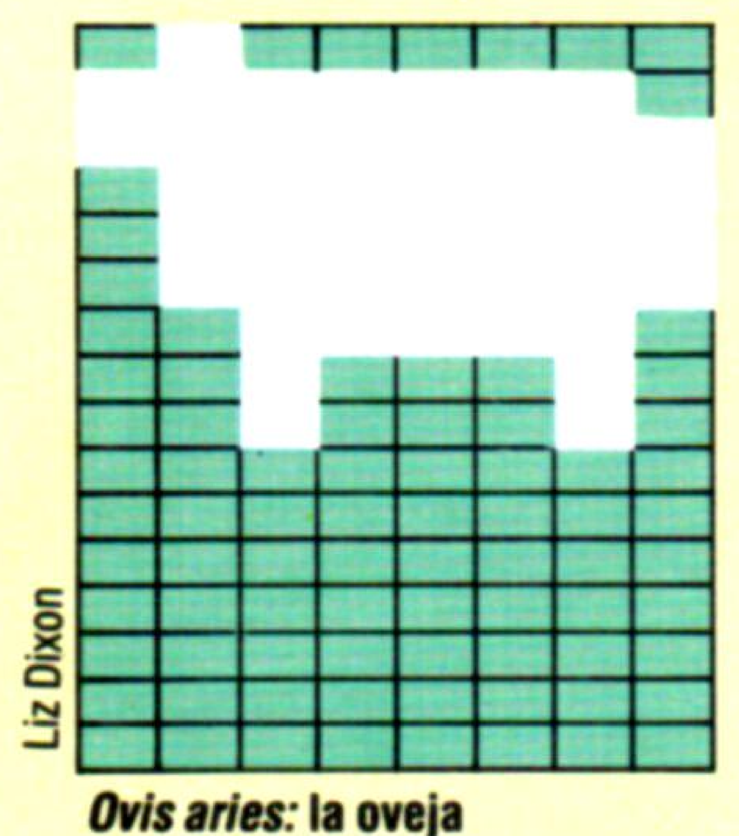
TO VIGILAR
  IF :SPEED=0[VERIFICAR]
  VIGILAR
END
```

En este procedimiento, **SPEED** da el valor de la velocidad de la tortuga en curso. El procedimiento **VERIFICAR** debe determinar cuál es el evento que se ha producido, llevar a cabo las acciones necesarias y después restaurar las velocidades. En este caso sólo estamos interesados en un evento, pero ilustra la forma de programar este método.

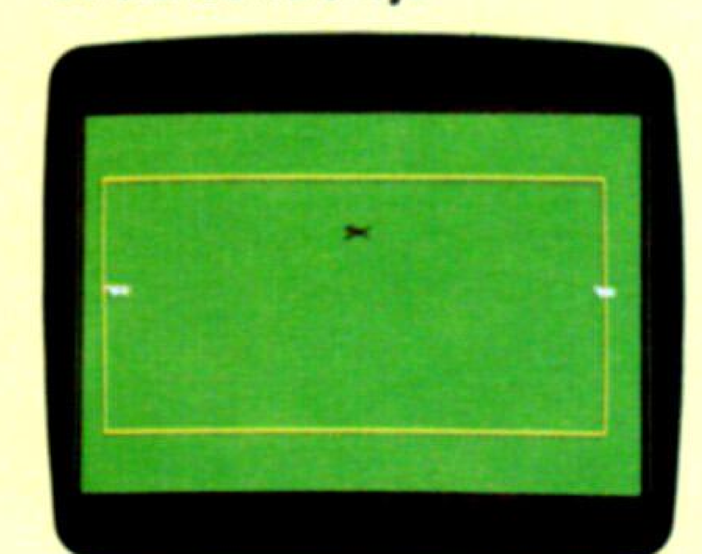
```
TO VERIFICAR
  IF COND OVER 0 0 THEN[TURN]
  SETSP 50
END
```

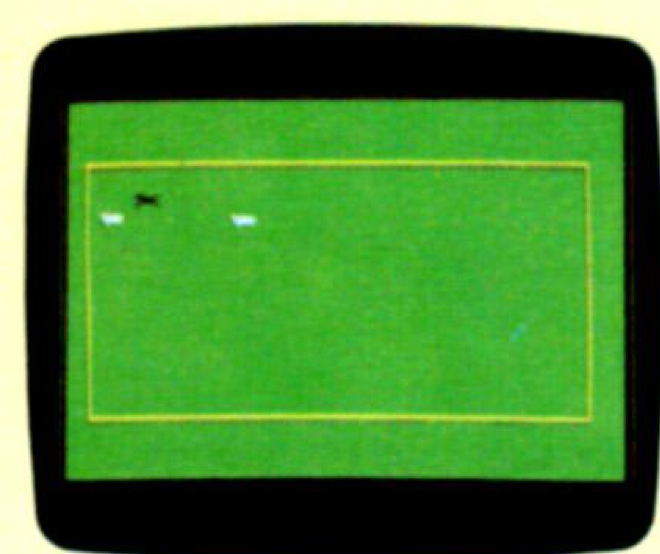
La instrucción **COND** y un número da una salida verdadera *(true)* si se ha producido un evento de ese número. **COND** sólo puede comprobar un evento en el momento en que el LOGO ejecuta la línea que lo contiene.


Canis familiaris: el perro


Ovis aries: la oveja

Liz Dixon


Empieza el juego


El juego en desarrollo


¡Otra vez en el redil!

Ian McKinnell

Agrupando ovejas

He aquí un juego que utiliza muchas de las características que hemos descrito. El jugador emplea la palanca de mando para controlar a un perro que está persiguiendo a dos ovejas por un campo. Si las ovejas tocan la cerca, retrocederán y girarán. Si chocan entre sí, girarán al azar. Si el perro toca a las ovejas, éstas girarán 90º hacia la derecha. Pulsando el botón de la palanca se dibuja una pequeña jaula en el rincón inferior izquierdo del campo. Volviendo a pulsar el botón la jaula se borra. La tarea del perro consiste en conducir a las ovejas hasta el interior de la jaula.

```
TO PERSEGUIR
   EST.VAR
   ASK :TORTUGA[EST.PANTALLA]
   EST.DIABLOS
   EMPEZAR
   VIGILAR
END

TO EST.VAR
   MAKE"CERCA 0
   MAKE"TORTUGA 0
   MAKE"OVEJA1 3
   MAKE"OVEJA2 2
   MAKE"PERRO 1
   MAKE"VERDE 92
   MAKE"NARANJA 23
   MAKE"NEGRO 0
   MAKE"BLANCO 7
END

TO EST.PANTALLA
   CS
   FS
   SETBG :VERDE
   HT
   PU
   SETPOS [-150 -80]
   PD
   SETPC 0 :MARRON
   RECT 160 300
   PU
END

TO RECT :LADO1 :LADO2
   REPEAT 2[FD :LADO1 RT 90 FD :LADO2 RT 90]
END

TO EST.DIABLOS
   WHEN OVER :OVEJA1 :CERCA[SETSP 0]
   WHEN OVER :OVEJA2 :CERCA[SETSP 0]
   WHEN TOUCHING :OVEJA1 :OVEJA2[SETSP 0]
   WHEN TOUCHING :PERRO :OVEJA1 [SETSP 0]
   WHEN TOUCHING :PERRO :OVEJA2 [SETSP 0]
   WHEN 3[SETSP 0]
   WHEN 15[JOYH]
END

TO JOYH
   IF (JOY 1)<0[STOP]
   ASK :PERRO[SETH 45*JOY 1]
END

TO EMPEZAR
   EST :OVEJA1 1[-150 20] 45 :BLANCO
   EST :OVEJA2 1 [150 20] 315 :BLANCO
   EST :PERRO 2[0 0]0 :NEGRO
   EST.VELOCIDADES
END

TO EST :NO :FORMA :POS :ENCABEZ :COLOR
   TELL :NO
   PU
   SETSH :FORMA
   SETC :COLOR
   ST
   SETPOS :POS
   SETH :ENCABEZ
END

TO EST.VELOCIDADES
   ASK :OVEJA1[SETSP 10]
   ASK :OVEJA2[SETSP 10]
   ASK :PERRO[SETSP 60]
END

TO VIGILAR
   IF SPEED=0[VERIFICAR]
   VIGILAR
END

TO VERIFICAR
   IF COND OVER :OVEJA1 :CERCA[ASK :OVEJA1
   [BK 20 RT 90]]
   IF COND OVER :OVEJA2 :CERCA[ASK :OVEJA2
   [BK 20 RT 90]]
   IF COND TOUCHING :OVEJA1 :OVEJA2[CHOCAR]
   IF COND TOUCHING :PERRO :OVEJA1[ASK
   :OVEJA1[RT 90]]
   IF COND TOUCHING :PERRO :OVEJA2[ASK
   :OVEJA2[RT 90]]
   IF COND 3[ASK :TORTUGA[DIBUJAR.JAULA]]
   EST.VELOCIDADES
END

TO CHOCAR
   ASK :OVEJA1[SETH RANDOM 360]
   ASK :OVEJA2[SETH RANDOM 360]
END

TO DIBUJAR.JAULA
   PU
   SETPOS[-150 -30]
   PX
   SETH 90
   REPEAT 2[FD 50 RT 90]
   PU
END
```

Ejercicios de LOGO

1. Modificar el juego de agrupar las ovejas de modo que el perro se controle utilizando el teclado en vez de la palanca de mando.
2. Escribir un programa para un juego en el cual usted está al mando de una nave espacial. Los meteoritos se precipitan hacia usted, y debe apartarse de su camino y sobrevivir el mayor tiempo posible. Le brindamos algunas pistas para ayudarlo. Utilice un sprite para la nave y los otros para los meteoritos. Emplee diablos WHEN para verificar las colisiones. Los meteoritos se mueven a una velocidad constante pero al azar. La nave se puede controlar mediante la palanca de mando.

Un digno sucesor

El Plus/4 supera a su antecesor, el Commodore 64, en varios aspectos: BASIC, memoria y programas incorporados

Sus fabricantes afirman que el Commodore Plus/4 se venderá al mismo tiempo que el Commodore 64, al cual no pretende sustituir. Pero el nuevo modelo ofrece tantas mejoras respecto al 64, que si consigue éxito en el mercado desplazará por completo a su antecesor.

El Plus/4 utiliza el microprocesador 7501, que es un desarrollo del 6502. Este chip está diseñado de tal modo que puede acceder a más de 64 K de memoria. Esto significa que la máquina tiene espacio para un BASIC muy satisfactorio, conservando al mismo tiempo su RAM libre para el usuario. Hay 64 K libres para utilizar con programas en BASIC, si bien esta cantidad se reduce a 50 cuando se emplean gráficos. Esto es más de lo que posee cualquier otro micro personal, a excepción del Sinclair QL (véase p. 981) y el Advance 86a (véase p. 829).

En el Plus/4 se ha implementado una versión excelente del BASIC Microsoft, y son particularmente dignas de mención las instrucciones para gráficos y sonido. En la modalidad de gráficos, la instrucción DRAW genera puntos o líneas y mediante la instrucción PAINT se puede rellenar con color cualquier forma dibujada. La instrucción BOX dibuja cuadrados y rectángulos sólo con líneas o con color sólido. La instrucción CIRCLE es particularmente versátil. Además de dibujar círculos, se pueden crear óvalos especificando la altura y la anchura del óvalo. Se pueden dibujar partes de óvalos para producir arcos, simplemente especificando en la instrucción posiciones de principio y final.

Todas las instrucciones operan normalmente en una pantalla con una resolución de 320×200 puntos. Esta resolución es la misma que en el Commodore 64, pero el Plus/4 realmente sobresale por su selección de colores. Puede mostrar 120 colores distintos en la pantalla al mismo tiempo, además del negro. Los mismos se crean a partir de 15 tonalidades básicas, cada una de las cuales se puede visualizar en ocho niveles diferentes de brillantez. Lamentablemente, el Plus/4 no puede producir sprites.

Las instrucciones para controlar el sonido son bastante estándares. La instrucción SOUND toca una nota de altura y duración especificadas. Una instrucción separada, VOL, especifica uno de ocho ajustes para el volumen de cada canal de sonido. Todo el sonido se emite a través del altavoz del televisor. El Plus/4 sólo proporciona dos canales de sonido, si bien el BASIC permite especificar tres. Este "tercer" canal es, en realidad, una facilidad de ruido y cualquier nota a la que se otorgue esa referencia de canal se reproducirá como tal. Esto es muy útil para los juegos, en los cuales se requieren efectos sonoros especiales.

Se han incluido instrucciones para mejorar el cuerpo principal del BASIC. Una instrucción AUTO producirá automáticamente números de línea en la introducción de programas; RENUMBER renumerará los números de línea de los programas, y VERIFY verificará que los programas se hayan salvado correctamente en cassette o en disco. Hay muchas instrucciones nuevas para trabajar con discos, y Commodore evidentemente desea venderles unidades de disco a un elevado número de usuarios de su nuevo ordenador.

El Plus/4 tiene una visualización de textos de 40×25 caracteres. El usuario puede especificar dos puntos de la pantalla para que actúen como las esquinas de una "ventana". Todo el texto, como listados e instrucciones, aparecerá entonces dentro de esa ventana, sin alterar el resto de la pantalla.

Las teclas del Plus/4 son muy sensibles al tacto, requiriendo sólo una mínima presión para que sean registradas. A numerosos caracteres, tales como @, =, +, − y £ se les asignan teclas propias y se puede producir un juego completo de caracteres para gráficos desde el teclado. En la parte superior de éste hay cuatro teclas de función y cuando se enciende la máquina éstas quedan preparadas para producir las instrucciones utilizadas más comúnmente. Las teclas de función permiten que el usuario defina una instrucción nueva de hasta 128 caracteres para cada tecla. A partir de las cuatro teclas se pueden producir ocho funciones diferentes empleándolas con la tecla de cambio (Shift).

La generosa cantidad de espacio de memoria permite que el Plus/4 posea software de aplicaciones incorporado. Se suministran cuatro programas: un paquete de tratamiento de textos, hoja electrónica, base de datos y gráficos. Estos programas



Chris Stevens

Apostando al futuro
El sucesor de Commodore, largo tiempo esperado, para el Commodore 64, posee todas las características que se están volviendo estándares en los micros más recientes: apariencia tipo MSX y racimo para cursor, memoria grande y software incorporado. Sin embargo, la competencia de ventas es intensa y los potenciales compradores están más informados que nunca. Commodore no está dando por segura su preeminencia en el mercado, tal como reflejan claramente el aspecto y las características del Plus/4



QL contra Plus/4
Objetivamente, no cabe ninguna comparación: el QL, con sus microdrives incorporados, su mayor memoria, su SuperBASIC y su excelente software es, de acuerdo a los estándares del mercado, una buena adquisición; el Plus/4, con grabadora de cassette, se destaca sólo por la calidad de su teclado. Una apreciación objetiva se inclinaría sin vacilar por el QL; sin embargo, es posible que los usuarios adopten una decisión basada en la fidelidad a una marca de prestigio

Ian McKinnell

Bus en serie
Aquí se pueden enchufar periféricos Commodore estándares, como una unidad de disco y una impresora

Conector para cassette
La grabadora de cassette exclusiva se enchufa aquí

Puerta para el usuario

Conectores para palanca de mando
Reciben las palancas exclusivas del Plus/4. No se pueden utilizar palancas estándares

Conector para salida de video y sonido

Modulador de TV
Proporciona una señal para un televisor normal

Carcasa ULA
Es metálica; contiene en su interior, para protegerlo de las interferencias de radio, un enorme chip ULA (matriz lógica de propósito general)

Unidades de ROM
Contienen el BASIC y los cuatro paquetes de software

están diseñados para trabajar de forma conjunta.

Lamentablemente, el procesador de textos es bastante difícil de utilizar. El Plus/4 sólo puede visualizar 40 caracteres en una línea de su pantalla, pero muchas impresoras pueden imprimir 80 caracteres por línea. Para adaptarse a esta anchura, el contenido de la pantalla se desplaza al llegar a la columna 37 y sigue desplazándose hasta alcanzar la columna 77; entonces vuelve a saltar a la primera columna. El programa posee instrucciones de formato para establecer márgenes y justificar texto, pero sólo entran en vigor cuando se imprime el texto. Asimismo, posee instrucciones SEARCH y REPLACE para localizar frases o palabras determinadas dentro de un documento y reemplazarlas si fuera necesario. El máximo de texto que se puede entrar es de 99 líneas. Con los 77 caracteres de tamaño de línea esto representa un máximo de 1 500 palabras, lo cual no es suficiente para aplicaciones serias.

El programa de hoja electrónica es más fácil de utilizar que el de tratamiento de textos, si bien sufre, asimismo, de las limitaciones que supone una visualización en pantalla de 40 columnas. Ello significa que sólo puede mostrar tres celdas de la hoja electrónica a lo ancho de la pantalla y 12 a lo largo, aun cuando le es posible manejar modelos de hasta 17 celdas de ancho por 50 de largo.

El programa para gráficos es más bien decepcionante. Todo lo que hace es transformar un conjunto de cifras de la hoja electrónica en una especie de torpe gráfico de barras, formado a partir de gráficos de bloques, y transferirlo al procesador de textos. Mediante el mismo se lo puede visualizar o imprimir.

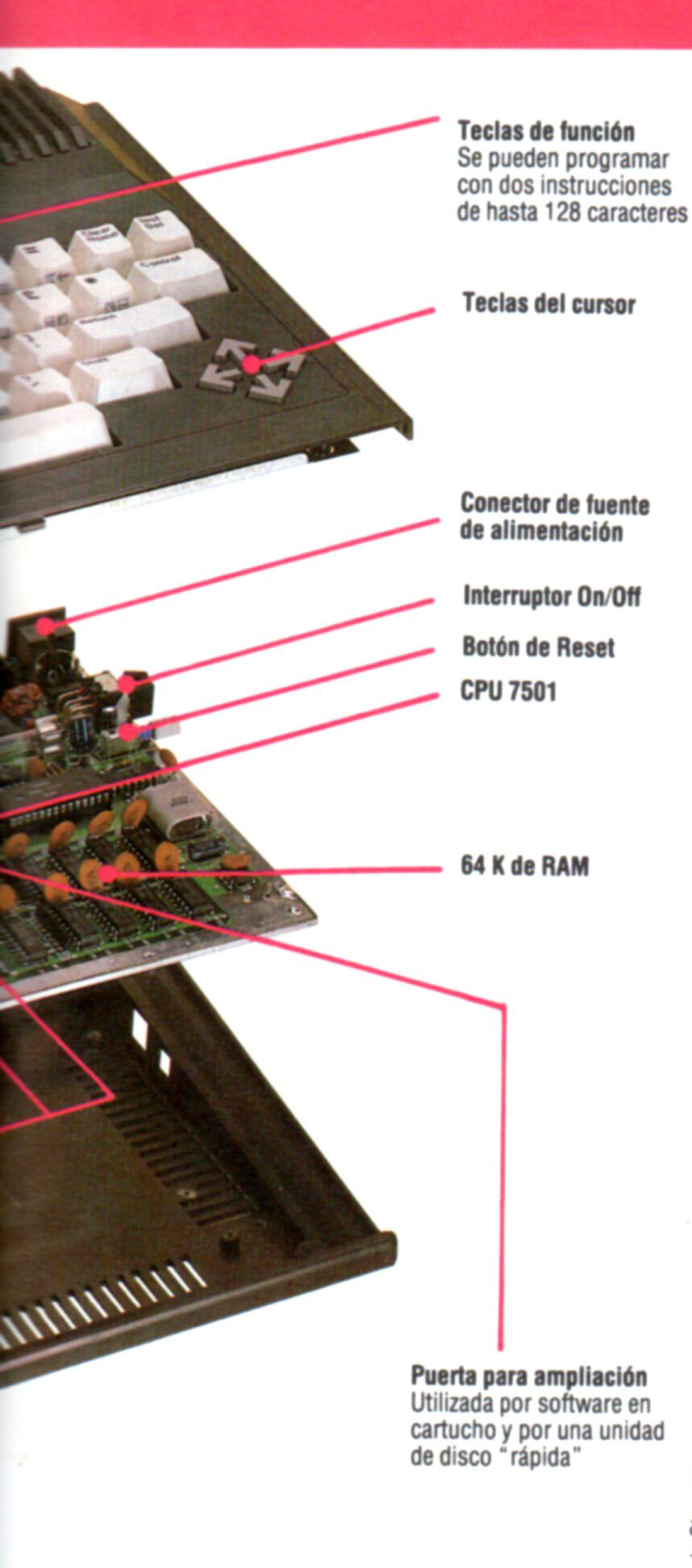
Tanto el procesador de textos como la hoja electrónica se pueden utilizar con la máquina estándar, pero la única forma de guardar sus resultados es disponiendo de una unidad de disco, lo cual significa que son de poca utilidad para las personas que dependan del almacenamiento en cassette. El último programa, el de base de datos, requiere una unidad de disco para su uso. Trabaja definiendo un formato estándar que se graba en disco como registros vacíos. Todos los datos se entran luego en los registros vacíos. Esto implica que un disco sólo se puede utilizar para una base de datos y el formato de la información no se puede modificar una vez que se han grabado datos. Cada base de datos puede contener hasta 999 registros, cada uno de ellos con un máximo de 17 campos de 38 caracteres. En líneas generales, el software es decepcionante: además de ser demasiado tosco para un uso de gestión serio, exige que el usuario personal adquiera una unidad de disco.

Muchos usuarios personales se sentirán más satisfechos con el monitor de código máquina incorporado, Tedmon, que con el software. El monitor es una gran ayuda para el programador de lenguaje máquina y se pone en funcionamiento mediante la instrucción MONITOR.

Commodore está produciendo numerosos accesorios para el Plus/4. Para muchos usuarios, el más importante de ellos será la grabadora de cassette. Al igual que otros micros Commodore, el Plus/4 requiere una grabadora fabricada especialmente por Commodore. Ésta emplea un enchufe diferente al de las más antiguas, por lo cual es necesario adquirir una grabadora nueva. El Plus/4 utiliza, asi-









Teclas de función

Racimo del cursor

Botón de Reset (puesta a cero)

Puerta para ampliación/Puerta para el usuario

Nuevas características
A quienes ya poseen experiencia en máquinas Commodore les interesará saber que el Plus/4 posee tanto tecla Escape como Reset. Otros incentivos son el racimo de las teclas para el cursor, las teclas de función (programables desde BASIC) y Help. Los conectores para la fuente de alimentación, para cassette y para palanca de mando son diferentes de los del C64 y el Vic-20, al igual que las puertas para ampliación y para el usuario

Chris Stevens



COMMODORE PLUS/4

DIMENSIONES
67 × 203 × 338 mm

CPU
7501, 0,9 o 1,8 MHz

MEMORIA
64 K de RAM, 64 K de ROM

PANTALLA
Texto: 40 × 25. Gráficos: 320 × 200 en 121 colores

INTERFACES
2 conectores para palanca de mando, interface en serie, interface para cassette, puerta para cartucho/en paralelo

LENGUAJES DISPONIBLES
BASIC

TECLADO
Tipo máquina de escribir con 67 teclas (cuatro son de función)

DOCUMENTACION
Manuales bien concebidos que enseñan a programar y a utilizar el software incorporado

VENTAJAS
Versión de BASIC particularmente acertada, con buenas instrucciones para gráficos y facilidades de estructuración. 64 K completos de memoria

DESVENTAJAS
Requiere grabadora de cassette exclusiva y palancas de mando no estándares. El software incorporado es pobre



Un sobrecosto oculto

El software basado en ROM del Plus/4 comprende un procesador de textos, una hoja electrónica, una base de datos y una utilidad para gráficos. El principal punto fuerte de los paquetes es el hecho de estar incorporados; sin embargo, no se pueden emplear sin unidad de disco, lo que incrementa sustancialmente el costo de la máquina

Ian McKinnell

mismo, un conector diferente para palancas de mando.

Se está ofreciendo para la máquina una versión ligeramente mejorada de la lenta unidad de disco Commodore. La firma también está desarrollando una unidad de disco "rápida" que se enchufará en la puerta para cartuchos de la máquina, para proporcionar velocidades más cercanas a las de las unidades normales.

Son al menos cinco las impresoras que funcionarán con el micro: una de rueda margarita, dos matriciales comunes, una matricial en color y una impresora/plotter de cuatro lápices.

El Commodore Plus/4 se está vendiendo a un precio ligeramente más alto que el 64, pero su BASIC mejorado y su espacio de memoria extra hacen que represente una buena compra. En un futuro capítulo analizaremos con detalle cómo trabaja esta nueva versión del BASIC Commodore. La falta de software seguirá siendo un problema hasta que la máquina consiga establecerse en el mercado. Pero una empresa con las cifras de ventas de Commodore no debería tener ningún problema para conseguir un sólido apoyo de software por parte de las firmas especializadas más importantes.

Clones en acción

Ahora veremos cómo utilizar las características del "Vu-Calc" para efectuar cálculos de pagos de hipotecas o préstamos bancarios

El gran poder de los programas de hoja electrónica, incluso de un paquete tan sencillo como el *Vu-Calc* de Psion, es la forma en que permiten aplicar fórmulas complejas a los datos. Como veremos, con el *Vu-Calc* es posible construir algunos modelos interesantes y útiles, a pesar del hecho de que este programa en particular proporciona muy poco en cuanto a fórmulas incorporadas. En realidad, la única fórmula incorporada es su capacidad para "sumar" (es decir, sumar entre sí los contenidos de) bloques de celdas; ésta se indica colocando un signo @ delante de la dirección de la celda.

Las hojas electrónicas más perfeccionadas contienen fórmulas incorporadas muy sofisticadas, que el usuario puede llamar por su nombre. La ventaja de un sistema de este tipo es que realmente no hace falta conocer cómo trabajan estas fórmulas. Si se desea utilizar una fórmula de hipoteca con el *Multiplan*, por ejemplo, para calcular los pagos previstos por la compra de una vivienda en distintos períodos de pago (supongamos, 15, 20 y 25 años), simplemente se llama a la fórmula y se entran los datos correspondientes. El *Multiplan* calcula entonces todas las respuestas.

Con el *Vu-Calc* estos mismos cálculos pueden exigir mucho más tiempo y esfuerzo. El usuario debe construir por sí mismo las fórmulas necesarias y luego entrarlas en la máquina. El *Vu-Calc* también le impone al usuario una serie de limitaciones. Posee un máximo de 28 columnas, de modo que el modelo más grande que se puede construir, con cada columna representando un mes, cubrirá un período de poco más de dos años. La exactitud también puede representar un problema; el *Vu-Calc* trabaja sólo con valores enteros (de números enteros) e ignora las cifras que aparezcan después de una coma decimal, de modo que 99,9 se tomará como 99. No obstante, sí permite entrar valores y operaciones aritméticas en cualquier lugar del modelo. Por ejemplo, si el cursor está situado en una celda vacía (H5, p. ej.), se puede entrar 500*2 en la línea de instrucción. A continuación, si se pulsa la tecla Enter, se visualizará el resultado correspondiente (1 000) en la celda H5.

Otra característica negativa del *Vu-Calc* es el modo en que se editan las fórmulas. Un paquete "elegante", como el *Lotus 1-2-3*, utiliza para la edición una tecla de función. La pulsación de la tecla coloca automáticamente el contenido de la celda que contiene el cursor en la línea de instrucción. *Vu-Calc* posee una instrucción EDIT (#E) que se emplea para modificar una fórmula, pero la fórmula se debe volver a digitar cada vez que se utiliza la facilidad de edición. Si está trabajando con una fórmula larga y se da cuenta de que se ha olvidado entrar un paréntesis, no existe modo alguno de insertarlo: es inevitable volver a digitar toda la línea. Lo único que hace la instrucción EDIT es decirle al programa que borre la fórmula antigua de una celda e inserte luego la nueva.

Sin embargo, la utilización de la instrucción REPLICATE (#R) con una fórmula permite realizar un modelado bastante más complejo. Vamos a suponer que se desea ampliar el ejemplo del presupuesto doméstico (véase p. 1172) para anticipar los aumentos inflacionarios del presupuesto doméstico de alimentos, dando por sentada una tasa de inflación uniforme del 0,5 % mensual. Efectuar los cálculos necesarios con lápiz y papel sería una tarea que, a todas luces, consumiría mucho tiempo. Con el paquete *Vu-Calc* se puede realizar rápidamente mediante el empleo de una fórmula y REPLICATE.

Para llevar a cabo la operación deseada, debe indicarse al *Vu-Calc* que incremente su presupuesto mensual inicial (p. ej., 20 000 ptas) en un 0,5 por ciento. Las hojas electrónicas más sofisticadas hacen que esto resulte sencillo, mediante la utilización de una instrucción GROW BY (incrementar en un), pero el *Vu-Calc* requiere que el usuario entre las operaciones aritméticas que se deben llevar a cabo. Con el objeto de que el programa reconozca una fórmula que contiene direcciones de celdas, la fórmula debe ir precedida con un $ o un %. Éstos son dos símbolos elegidos de forma arbitraria que no tienen ninguna relación ni con dólares ni con porcentajes, sino que le dicen al programa que las direcciones de celdas son significativas en la fórmula que se está considerando, y estas direcciones pueden ser relativas (%) o bien absolutas ($). Una referencia a una celda absoluta le dice al *Vu-Calc* que busque y actúe sobre un valor de una celda, independientemente de la posición de ésta.

Para ver lo que hace una "dirección relativa", retomemos nuestro ejemplo. La fórmula para "incrementar" el presupuesto en un 0,5 por ciento es %B3*100,5/100, donde % indica una dirección de celda relativa y B3 es la dirección de la celda que contiene el valor que representa el presupuesto mensual para alimentos. Habiendo digitado esta fórmula en la celda B4, hemos entonces de copiar la fórmula para obtener el resultado para todo el año. B4 visualizará el resultado numérico de la fórmula; la fórmula propiamente dicha aparece en la parte inferior de la hoja de trabajo cuando el cursor está en B4. La instrucción REPLICATE #R,B4,B5:B14 nos da el resultado deseado (B4 contiene la fórmula, B5:B14 define la serie de celdas a través de las cuales se produce la reproducción). Los resultados se muestran casi al instante y nuestro modelo de hoja electrónica tendrá el siguiente aspecto:

	1	2	3	4	5
A			ENE	FEB	MAR
B	Pres. alim.		20 000	20 100	20 200

La visualización muestra por qué se utiliza la celda B3 para retener la cifra del presupuesto mensual inicial: la etiqueta se extiende a través de las columnas uno y dos, de modo que empezamos en la columna tres para crear una visualización pulcra. Observe que todas las cifras son valores enteros; la cifra de marzo debería ser 20 200,5 pero la hoja electrónica "redondea", de modo que la misma se visualiza exactamente como 20 200. La cifra de abril en realidad sería 20 301,502, pero *Vu-Calc* la tomará como 20 301. Dado que el aumento inflacionario se va haciendo mayor mes a mes, del mismo modo la discrepancia entre el valor real y la cifra visualizada se irá volviendo también mayor.

Este sencillo ejemplo demuestra el efecto de la instrucción REPLICATE cuando se utiliza con direcciones de celdas relativas. Cada vez que el programa escribe la fórmula en la siguiente celda por la derecha, la fórmula cambia de acuerdo con esto. Nuestra fórmula original en B4 era %B3*100,5/100. Esta fórmula se reproduce en B5 como %B4*100,5/100, en B6 como %B5*100,5/100, y así sucesivamente. En cada caso, el número de columna de la dirección de celda se incrementa en razón de uno. La reproducción a través de una columna hacia abajo produce el mismo efecto en las direcciones (es decir, E1 se convierte en F1, etc.). Si en vez de direcciones relativas hubiésemos utilizado direcciones absolutas ($), este "cambio" en las direcciones de celdas no se habría producido, sino que a través de todas las celdas se habría reproducido la misma fórmula y el valor en cada una de ellas sería idéntico al valor mostrado en B4.

Intentemos ahora emplear el mismo modelo para predecir los gastos mensuales realizados por una empresa en materias primas, empezando con 10 000 000 de ptas por mes e incrementando en un 0,5 % mensual durante dos años. ¿Cuánto más costarían las mercancías si se las comprara a mediados del segundo año? Utilizando nuestro modelo, esto se puede calcular muy rápidamente.

Cambie el valor en B3 por 10 000 000, desplazando el cursor hasta B3 y digitando la nueva cifra. Ahora utilice la instrucción REPLICATE para extender la fórmula desde B14 hasta B26, para conformar los 24 meses completos. Las hojas electrónicas más sofisticadas mostrarán los nuevos resultados en el momento en que se cambie el valor de B3. No obstante, con el *Vu-Calc* se deben volver a calcular los resultados (que por el momento aún se basan en la fórmula antigua) mediante el empleo de la instrucción CALCULATE, #C. El programa calcula entonces los nuevos valores y visualiza la respuesta que requerimos en la celda B20. Si prueba este ejemplo, verá que la cantidad es 10 993 100 ptas: un incremento de casi un millón de pesetas. Ésta no es una cifra exacta, puesto que todos los números se redondean, pero es suficientemente aproximada como para darle una idea del efecto de la inflación durante este período.

A modo de último ejemplo del tipo de problemas de una sola fila, vamos a tomar una fórmula más complicada, diseñada para calcular la reducción de saldo de una deuda de tarjeta de crédito o préstamo bancario de 100 000 ptas, sobre la cual se está pagando un interés del 27 % anual. Suponiendo que usted esté devolviendo 8 000 pesetas al mes, ¿cuándo lo terminará de pagar? La información necesaria para calcular esto está en el principal del préstamo, más el interés para el mes, menos el pago mensual. De modo que si digitamos 100 000 en B1, la fórmula será %B1+%B1*,27/12-80. Reproduzca la fórmula a través de las 28 columnas del modelo, explore la fila para hallar el punto en el cual la cantidad se vuelve positiva, y habrá hallado el punto en el cual el saldo estará pagado por completo y usted estará con saldo a su favor en caso de continuar los pagos mensuales. De acuerdo a nuestro modelo, esto ocuparía 16 meses. A modo de complemento, también posee una pulcra visualización de su saldo pendiente mes a mes, suponiendo que mantenga constantes sus pagos de 8 000 ptas.

Relativamente absolutas

En este simple modelo hemos utilizado la instrucción REPLICATE a través de las filas C, D y E; para claridad del ejemplo, reproducimos la fórmula en cada celda. La celda C3 se ha reproducido de forma absoluta a todo lo largo de la fila, de modo que en todas las celdas C aparece la misma fórmula, A5/12, con el mismo resultado: 45 000 ptas. Las fórmulas de las celdas D4 y E3, sin embargo, se han reproducido de forma relativa, por lo cual todas las referencias a celdas de las fórmulas cambian de una celda a otra a lo largo de la fila, con resultados consiguientemente variables

	1	2	3	4	5
A		INGRESOS ANUALES=540 000			
B			ENERO	FEBRERO	MARZO
C	INGRESOS DOMÉSTICOS		A5/12 45 000	A5/12 45 000	A5/12 45 000
D	GASTOS DOMÉSTICOS		DATOS 40 000	D3*1,01 40 400	D4*1,01 40 800
E	SALDO MENSUAL		C3-D3 5 000	C4-D4 4 600	C5-D5 4 200

Liz Dixon

Control de energía

Esta vez construiremos un convertidor de digital a analógico para añadir a nuestro sistema para la puerta para el usuario

Para este proyecto hemos optado por utilizar un convertidor de digital a analógico ya hecho en un chip, si bien se puede construir un circuito a partir de distintos componentes. La salida analógica de este chip, el DAC, se deposita con un amplificador en un segundo chip. La salida de éste se conduce directamente a una salida y a través de un condensador y un control de nivel a la otra salida.

Paso uno: Corte la carcasa para acomodar las dos conexiones del bus del sistema. También se puede cortar un conector de empalme para utilizarlo en futuros proyectos.

Paso dos: Corte la veroboard (de 30 agujeros por 16 franjas). Ahora haga los cortes de pistas tal como indica el diagrama. Primero suelde en su lugar los conectores de chip, luego los cables de enlace. A continuación se deben colocar en su lugar los dos condensadores. No tiene importancia de qué lado se instalen. Si desea instalar el conector para ampliación del bus, entonces suéldelo en su sitio ahora e instale el cable plano.

Paso tres: Coloque los cuatro conectores en la carcasa, con el potenciómetro. Haga las conexiones entre éstos con el alambre estañado. Lleve los tres cables aéreos hasta la placa de circuitos.

Paso cuatro: Enchufe los dos chips y el convertidor estará ya completo. Observe que los dos chips no se enchufan con la misma orientación: el chip convertidor D/A se debe colocar de modo que la muesca quede a la izquierda cuando se lo mira desde arriba, con el conector macho del bus en la parte superior; la muesca del chip amplificador debe ir a la derecha.

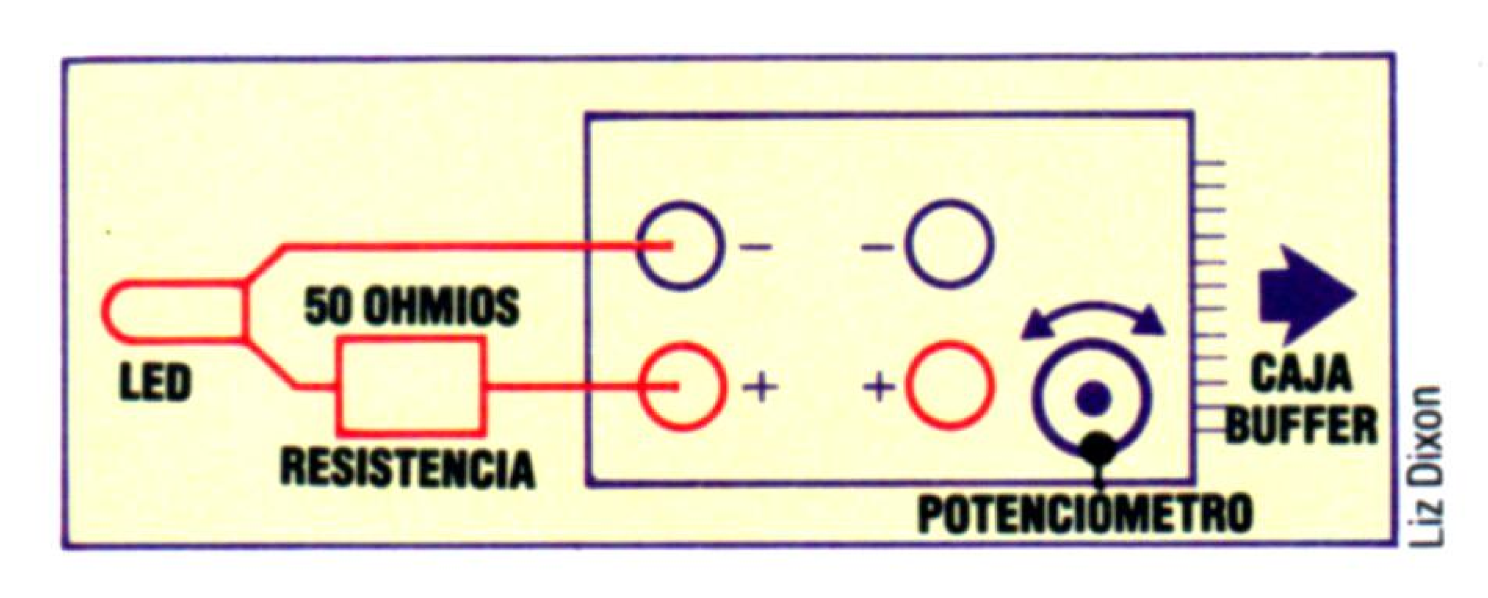
Después de haber construido el convertidor de digital a analógico y haber verificado cuidadosamente todas las conexiones, puede probar la unidad. El convertidor D/A convertirá en un voltaje cualquier valor binario de ocho bits que se coloque en el registro de datos de la puerta para el usuario. Este voltaje sale de la unidad de dos formas. En el par de conectores de salida de CD (corriente directa) se obtiene un voltaje de CD comprendido entre 0 y +2,5 V, correspondiente a los valores digitales de la puerta para el usuario de 0 a 255. El otro par de conectores de salida es para permitirnos simular una salida de CA (corriente alterna). El nivel de voltaje global se controla mediante un potenciómetro y se puede ajustar para adaptarlo a la entrada requerida por otro aparato.

Para probar la unidad, podemos desarrollar un experimento sencillo para alterar la brillantez de un LED. Para hacerlo se requieren estos pasos:

Paso uno: Conectar en serie un LED, del tipo utilizado en la caja buffer original (véase p. 1003), a una resistencia de 50 ohmios.

Paso dos: Conectar la unidad convertidora D/A directamente a la puerta para el usuario y proporcionarle energía de la forma habitual.

Paso tres: Conectar el LED y el circuito de la resistencia a través de los conectores de salida CD de la caja D/A y ejecutar este programa:




```
10 REM** PROGRAMA DE PRUEBA DE D/A PARA CBM 64 **
20 RDD=56579:REGDAT=56577
30 VL=127
40 POKE RDD,255:REM TODAS SALIDA
50 POKE REGDAT,VL
60 PRINT VL
70 GET A$
80 IF A$<>"Z" AND A$<>"X" THEN 70
90 IF A$="X" THEN DV=1
100 IF A$="Z" THEN DV=-1
110 VL=VL+DV
120 IF VL<256 AND VL>=0 THEN 50
```

```
10 REM**** PROGRAMA DE PRUEBA DE D/A PARA BBC ****
20 RDD=&FE62:REGDAT=&FE60
25 valor=127
30 ?RDD=255:REM TODAS SALIDA
40 REPEAT
55 ?REGDAT=valor
57 PRINTvalor
60 A$=GET$
62 IF A$<>"Z" AND A$<>"X" THEN 60
65 IF A$="X" THEN dv=1 ELSE dv=-1
67 valor=valor+dv
70 UNTIL (valor>255 OR valor<0)
```

Este sencillo programa dedica a salida las ocho líneas de la puerta para el usuario al colocar 255 (es decir, 11111111) en el registro de dirección de datos. En el registro de datos de la puerta para el usuario se coloca entonces un valor inicial de 127. Pulsando las teclas Z o X, el valor del registro de datos disminuye o bien aumenta respectivamente. El programa termina cuando el valor del registro cae fuera de la escala comprendida entre 0 y 255.

Incrementando el valor *digital* presente en el registro de datos podemos producir voltajes *analógicos* crecientes para el LED. A medida que el voltaje vaya aumentando hasta un nivel aceptable, el LED comenzará a brillar, primero tenuemente y luego con más intensidad a medida que se vaya aumentando el voltaje, hasta alcanzar una brillantez máxima cuando el valor presente en el registro de datos de la puerta para el usuario sea de 255.

Si su LED no se ilumina, intente invertir las conexiones a la caja convertidora D/A, antes de verificar la existencia de cualquier otro fallo. A diferencia de una bombilla normal, que se ilumina independientemente de en qué dirección fluya la corriente, un LED sólo se encenderá cuando ésta fluya en una determinada dirección.

Extraer el dígito
El convertidor de digital a analógico que vemos aquí incorpora un mando de potenciómetro excesivamente grande. Esto no es necesario, y debería arreglarse como para poder usar una perilla de control. Los lectores ya sabrán que cuando uno sale a comprar componentes electrónicos, con frecuencia tiene que quedarse con lo que consigue y adaptarlo a sus necesidades


Ian McKinnell



Circuito completo

Debe ser muy cuidadoso respecto a los detalles de trazado, en especial a la ubicación y limpieza de los cortes de pistas. Compruebe con regularidad su trabajo mediante el tester, inserte los componentes pasivos y los cables de enlace primero, utilice la soldadura discretamente y el estaño de soldar, y preste especial atención al posicionamiento de los chips

Lista de componentes

Cantidad	Artículo
1	ZN425 DAC
1	Amplificador CA3140
1	Conector DIL de 16 patillas
1	Conector DIL de 8 patillas
1	Condensador de 220 nF
1	Condensador de 0,47 µF
1	Potenciómetro rotativo de 10 K
2	Conector de 4 mm rojo
2	Conector de 4 mm negro
2	Enchufe de 4 mm rojo
2	Enchufe de 4 mm negro
1	Enchufe minicon de 12 vías
1	Conector minicon de 12 vías
	4 pulgadas de cable plano de 5 vías*
	Cable estañado pelado*
1	Veroboard de 50 agujeros×24 franjas
1	Caja de 80×61×41 mm

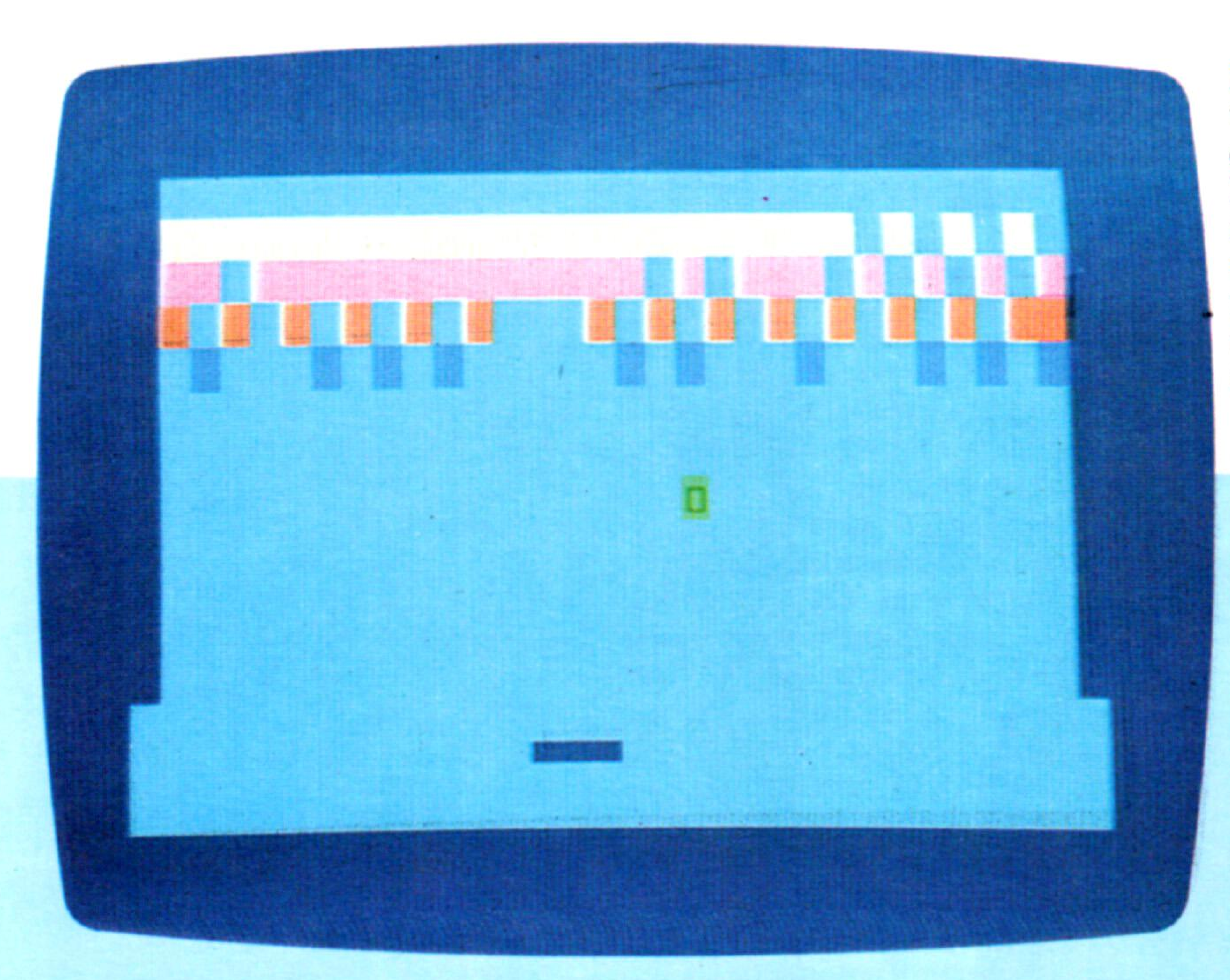
*Estos componentes ya los debe de tener como restos de proyectos anteriores.

Dentro de la caja

Kevin Jones

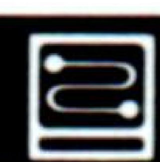
La pared

En esta ocasión presentamos el popular juego de “romper la pared”, tan habitual en las máquinas recreativas comerciales. He aquí la versión para el Dragon



El objetivo del juego es muy sencillo: intentar destruir una pared de ladrillos con ayuda de una pelota que debe ser relanzada con su raqueta. Cada ladrillo que rompa le proporciona un punto. Cuando el muro ha sido destruido por completo, aparece otro nuevo. El usuario dispone de diez pelotas para intentar conseguir el mayor número posible de puntos. Utilice la palanca de mando o las teclas A, S y la barra espaciadora para desplazar la raqueta.

```
10 REM **********************************
20 REM *              LA PARED          *
30 REM **********************************
40 S=0:NB=0
50 GOSUB 530
60 V=V+DV:H=H+DH
70 POKE B+1024,CN
80 B=V*32+H
90 L=PEEK (B+1024)
100 IF L<>223 AND L<>128 THEN DV
    =-DV:K=0:S=S+1
110 POKE B+1024,CB
120 IF V=13 AND ABS(R-29-B)>1THEN
    320
130 IF V=13 AND H>2 AND H<29 THEN
    POKE B+1024,CN:H=H+CH
140 IF V=13 OR V=1 THEN DV=-DV
150 IF H=1 OR H=30 THEN DH=-DH
160 ON JS GOSUB 220,250
170 IF R<446 THEN R=446
180 IF R>475 THEN R=475
190 PRINT@ R,R$;
200 IF S/120=INT (S/120)AND K=0
    THEN GOSUB 670
210 GOTO 60
220 L=INT (JOYSTK(0)/2+445)
230 IF ABS(L-R)>1 THEN R=R+2*SGN
    (L-R):CH=SGN(L-R)
240 RETURN
250 D$=INKEY$
260 D=2*((D$="A")-(D$="S"))
270 IF D$=" " THEN D0=0
280 IF D<>0 THEN D0=D
290 R=R+D0
300 CH=SGN(D0)
310 RETURN
320 NB=NB+1
330 FOR I=1 TO 5
340 SOUND 1,1
350 FOR J=1 TO 50
360 NEXT J,I
370 IF NB=11 THEN 410
380 POKE B+1024,CN
390 GOSUB 760
400 GOTO 60
410 IF S>R1 THEN R1=S
420 PRINT@ 166,"PUNTOS  :";S;
430 PRINT@ 230, "RECORD  :";R1;
440 PRINT@ 294, "OTRA ?";
450 FOR I=1 TO 100
460 D$=INKEY$
470 NEXT I
480 D$=INKEY$
490 IF D$="" THEN 480
500 IF D$<>"N" THEN 40
510 CLS
520 END
530 CLS
540 PRINT@ 203, "PAL.MANDO ?";
550 D$=INKEY$
560 IF D$="" THEN 550
570 IF D$="S" THEN JS=1 ELSE JS=2
580 CLS 6
590 R=461
600 FOR I=1024 TO 1055
610 POKE I,128
620 NEXT I
630 FOR I=1 TO 12
640 POKE I*32+1024,128
650 POKE I*32+1055,128
660 NEXT I
670 FOR I=1057 TO 1086
680 POKE I+32,159
690 POKE I+64,239
700 POKE I+96,255
710 POKE I+128,175
720 NEXT I
730 K=1
740 R$=CHR$(223)+CHR$(223)+CHR$
    (211)+CHR$(211)+CHR$(211)+CHR$
    (223)+CHR$(223)
750 CB=79:CN=223
760 V=13:DV=-1
770 H=RND(28)+1
780 B=V*32+H
790 DH=(RND(2)-1.5)*2
800 B1=B:D0=0
810 RETURN
```



Tránsito libre

En este capítulo del curso sobre el assembly del 6809 trataremos de las técnicas generales de programación en lenguaje máquina

Si un programa es escrito en código *reubicable* o *independiente de la posición*, éste puede alojarse en cualquier lugar de la memoria y ejecutarse sin más cambios. Lo cual es importante en los sistemas multiusuario y multitarea, en los que varios programas pueden estar ocupando la memoria al mismo tiempo, y para que el uso de la memoria sea eficiente, el sistema operativo deberá poder cargarlos en los lugares más convenientes. Hasta en los sistemas monousuario más sencillos suele tener su importancia la posibilidad de mantener bibliotecas de subrutinas y de construir un programa con módulos autosuficientes, en cuyo caso las rutinas pueden cargarse no siempre en el mismo sitio.

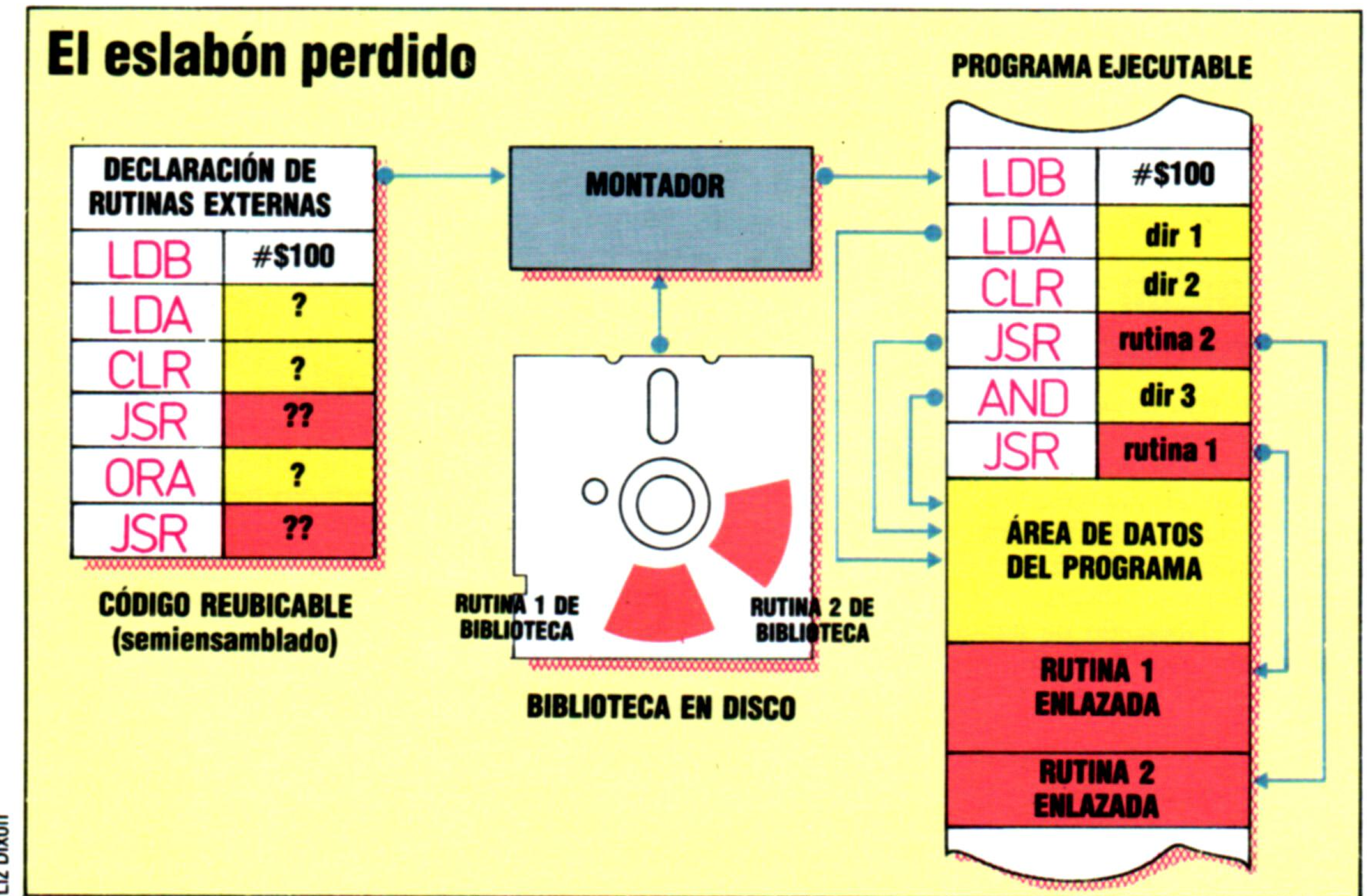
Muchos procesadores solventan la cuestión mediante un programa conocido como *montador (linking loader)*. El ensamblador elabora un código reubicable que prescinde de todas las referencias a direcciones absolutas de memoria; el montador pone en los lugares adecuados del programa las direcciones reales según va cargando el programa en la memoria. Y puesto que el montador mismo se ocupa de las direcciones, se asegura por completo la correcta transferencia de control entre los módulos. De este modo es posible escribir fragmentos de código en lenguajes diferentes que se compilen o ensamblen en el mismo código reubicable; así, por ejemplo, programas en PASCAL podrían llamar rutinas pertenecientes a una biblioteca en FORTRAN. El 6809 admite también esta solución, del todo necesaria cuando la construcción de programas es por módulos. El 6809 además facilita el proceso, pues permite la escritura en código totalmente reubicable de manera directa, por lo que no se necesita la fase adicional de insertar direcciones.

Un punto esencial en la escritura de código reubicable consiste en referenciar todas las direcciones mediante un *desplazamiento* (offset) relativo al contador del programa (PC). Un programa puede emplear una dirección de dos maneras: o como dato o como destino en una transferencia de control. Las instrucciones de bifurcación (BRA, BSR, etc.) calculan sus destinos en base a desplazamientos relativos al PC y son necesarias en toda transferencia de control dentro del programa del usuario. Las instrucciones de transferencia absoluta (JMP y JSR) sólo se emplearán para destinos que ocupan siempre el mismo lugar en la memoria, como ocurre con las rutinas del sistema operativo.

La tarea más difícil está en independizar todas las referencias a posiciones de datos. El 6809 lo consigue permitiendo la indexación mediante el PC. La instrucción:

```
LDA     OFFSET,PC
```

sumará ese offset o desplazamiento con signo al valor actual del PC para obtener la *dirección efectiva*. El problema reside en calcular el desplazamiento correctamente, para lo que es necesario hallar la diferencia entre la dirección de los datos y el valor



Montador
En los grandes sistemas, los programas en código máquina se cargan en la memoria mediante el montador *(linking loader)*. Esta utilidad del sistema operativo recoge el código semiensamblado (o sea, sin direcciones absolutas) del ensamblador, y determina cuál es la mejor dirección ORG para él dado el estado actual del sistema. Emplea esta dirección para sustituir las direcciones simbólicas que el ensamblador dejó en el programa sin dirección absoluta, para enlazar a continuación la rutina de biblioteca que el programador desea incorporar al programa; esta rutina (o rutinas) es cargada procedente de la biblioteca en disco y agregada al programa. Por último, el montador pasa el programa, para su ejecución, al sistema operativo

Liz Dixon

actual del PC, sin olvidar que el PC queda incrementado tan pronto como la instrucción a la que apunta es cargada en el procesador para su ejecución. Cuando una instrucción está siendo ejecutada, el PC queda, como se sabe, apuntando a la instrucción siguiente.

El procedimiento se complica a causa de la variación en las longitudes de las instrucciones en el 6809, que van de un byte hasta cinco. Por ejemplo:

```
LDX OFFSET,Y
```

emplea un byte para el opcode y otro byte para el *post-byte*, o sea, el byte empleado en cualquier instrucción indexada para especificar el registro índice a usar, independientemente de que se tenga en cuenta o no la indirección. El desplazamiento emplea dos, uno o ningún byte, según su tamaño. Los desplazamientos "cero" y los que pueden expresarse en cinco bits pueden ser incorporados en el post-byte (aunque esta opción no es muy bien manejada por algunos ensambladores). Desplazamientos más grandes necesitan un byte más (si pueden ser expresados en ocho bits) o dos bytes más. Cuando el PC es usado con indexación, los desplazamientos especiales de cinco bits o ninguno no son admisibles. La instrucción:

```
LDY OFFSET,X
```

necesita incluso un byte más, dado que el opcode de LDY es de dos bytes.

Si le agrada escribir en lenguaje assembly y está familiarizado con las decisiones de qué direcciones de datos usar y los lugares donde colocar las subrutinas, verá muy pronto que las tareas de manejar los opcodes, convertir las direcciones en formato de dos bytes y calcular desplazamientos de salto, le resultarán completamente naturales. Más simple que esta escritura en assembly manual es la adquisición de un ensamblador para que la realice él, porque en todo caso habrá de calcular la longitud de cada instrucción. Muchos ensambladores emplean la sigla especial PCR (Referido al Contador del Programa) para que el ensamblador haga uso del PC como un registro índice y calcule el desplazamiento. Por ejemplo:

```
DATITM     FCB  0
......................
           LDA  DATITM,PCR
```

Simulación de terminales

Damos en esta lección una subrutina que emplea esta técnica para la emulación de diversos terminales, de tal modo que el programa escrito para un determinado terminal pueda ser ejecutado también en otro sistema. Las diferencias entre terminales se hacen más evidentes en la codificación empleada para controlar las diversas funciones de pantalla, como son el borrado de pantalla y el posicionamiento del cursor. Los códigos pueden ser de control (caracteres cuyo ASCII es inferior a 32) o secuencias de escape, consistentes en el carácter de Escape (27 en ASCII) seguido de cualquier otro carácter o secuencia de caracteres. Nuestra sencilla rutina sólo nos permitirá la sustitución de un carácter de control por otro, o de un único carácter a continuación de un Escape por otro. Pero esta rutina es muy útil para mostrar cómo funciona una emulación. Hay dos tablas: una contiene los caracteres de control y la otra los caracteres de Escape. Si un programa genera un carácter de control, pongamos el caso, este carácter sirve de desplazamiento de la tabla de donde se sacará el carácter que ha de visualizarse.

Dado que la rutina es totalmente reubicable, la podemos añadir a cualquier programa y en cualquier posición. Suponemos que hay una rutina de sistema operativo (OUTCH) que envía el carácter contenido en el acumulador A a la pantalla, y empleamos JMP para acceder a esta rutina que ha de hallarse en una posición fija de la memoria. Se observará que falta por dar la directiva ORG, aunque no tenga efecto alguno. El carácter a visualizar debe encontrarse en A.

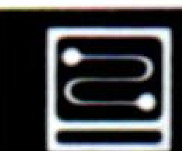
Longitud de las instrucciones

El problema de calcular la longitud de las instrucciones no se limita al empleo de un direccionamiento con PCR. A menudo es necesario conocer la longitud total de una rutina que debe ajustarse a un espacio limitado de la memoria, por ejemplo, en una ROM. Cualquier libro sobre el assembly del 6809, o cualquier manual del ensamblador, debe incluir una tabla de mnemotécnicos junto con sus datos asociados. Por cada expresión mnemotécnica, el dato incluirá el opcode correspondiente, la longitud total de la instrucción (aunque a veces esto no es posible, en cuyo caso la longitud mínima será seguida del signo +), el número de ciclos de reloj que emplea la instrucción al ser ejecutada, y el efecto de la instrucción sobre los flags del código de condición.

Veamos aquí las reglas generales para hallar la longitud de una instrucción, para así poder escribir en código compacto:

1) La mayoría de opcodes son de un solo byte; los que afectan directamente al contenido de S e Y (salvo en LEA) y algunos que afectan a U (como LDY y STS) son de dos bytes.
2) Todo direccionamiento indexado requiere un post-byte, y a veces uno o dos bytes más, según el tamaño del desplazamiento.
3) Los datos que van a continuación de un opcode en el modo inmediato son de uno o dos bytes, según el tamaño del registro empleado.
4) Las direcciones han de ser de un byte cuando se trata de la página directa (posiciones del $00 al $FF, generalmente) y de dos bytes en los restantes casos. No todos los ensambladores emplean apropiadamente el direccionamiento directo, causa por la cual una dirección requiere dos bytes cuando sería de esperar sólo uno.

Igualmente complejo es el problema del cálculo del tiempo empleado en la ejecución de cada instrucción, por la sencilla razón de que el tiempo depende a su vez de los bytes cuestionados, es decir, de la longitud de la instrucción. Pero es importante en aplicaciones en tiempo real y en el tratamiento de ciertos periféricos. El tiempo de cada instrucción se mide en ciclos de reloj (o al menos en el mínimo número de ciclos de reloj) que emplea la instrucción. Lo cual significa que el tiempo real em-



pleado depende también de la frecuencia de oscilación del reloj. La frecuencia de reloj más común para sistemas con 6809 es de 1 MHz por segundo (un millón de ciclos por segundo). O sea, cada ciclo tarda una millonésima de segundo. La instrucción inmediata:

```
LDA DATITM
```

que utiliza una dirección de 16 bits, emplea cinco ciclos, es decir, se ejecuta en cinco millonésimas de segundo. La instrucción:

```
PSHS PC,B,CC
```

emplea cinco ciclos, más otro ciclo por cada byte que pone en la pila. En este caso, el total es de nueve ciclos (recuerde que el contador del programa es de dos bytes).

Si un sistema no lleva incorporado un reloj en tiempo real, la única manera de medir el tiempo transcurrido será mediante una rutina software de demora. Tal rutina pone en funcionamiento una serie de instrucciones cuyos tiempos respectivos son conocidos de antemano, de tal modo que su suma proporciona el intervalo pedido. Tales intervalos suelen medirse en milisegundos (milésimas de segundo), por tanto no es necesario ser excesivamente precisos, pues un error de millonésima de segundo no es significativo. Suponiendo la velocidad a 1 MHz por segundo, la rutina de demora (Software Delay) que proporcionamos en esta página dará intervalos entre 1 y 255 milisegundos: el número exacto de milisegundos (ms) se coloca como un parámetro en A. La notación (A) significa el contenido del acumulador A.

Podemos expresar el cálculo de la constante COUNT como sigue:

Instrucción	Núm. de ciclos reloj	Número de veces ejecutadas	Tiempo empleado (ciclos de reloj)
PSHS B,CC	7	1	7
LDB #COUNT	2	(A)	(A)*2
DECB	2	(A)*COUNT	(A)*COUNT*2
BNE LOOP2	3	(A)*COUNT	(A)*COUNT*3
DECA	2	(A)	(A)*2
BNE LOOP1	3	(A)	(A)*3
PULS PC,B,CC	9	1	9

Lo que da un total de (A)*(7+5*COUNT)+16 ciclos de reloj. Para facilitar los cálculos, despreciaremos el 16. A un MHz por segundo, hay mil ciclos de reloj en un milisegundo, luego el tiempo total será (A)*1000 ciclos de reloj.

(A)*(7+5*COUNT) = (A)*1000
(7+5*COUNT) = 1000
5*COUNT = 973
COUNT = 195 (redondeado)

Es factible hallar intervalos más precisos y usar registros de 16 bits si es necesaria una mayor escala, pero el principio de decrementar un registro un número determinado de veces será el mismo.

Rutina de simulación de terminal

ESCAPE	EQU	27	
SPACE	EQU	32	(32: Espacio en ASCII)
OUTCH	EQU		Poner aquí la dirección del sistema operativo
	ORG	$1000	
CTABLE	RMB	32	Tabla de caracteres de control
ETABLE	RMB	128	Tabla de caracteres de Escape
EFLAG	FCB	0	Flag que indica si el último carácter fue un Escape
DISPCH	PSHS	X	Guarda X
	TST	EFLAG,PCR	Comprueba si el último carácter fue un Escape
	BEQ	DISP1	En caso negativo, va a DISP1
	LEAX	ETABLE,PCR	En caso afirmativo, carga la dirección de ETABLE en X
	LDA	A,X	Toma el carácter de sustitución por medio del carácter original contenido en A considerado como desplazamiento
	CLR	EFLAG,PCR	Restablece EFLAG
	BRA	FINISH	
DISP1	CMPA	SPACE	Carácter de control?
	BGE	FINISH	No lo es : va a FINISH
	CMPA	ESCAPE	Sí lo es : comprueba si es Escape
	BEQ	ESCCH	Caso de serlo, va a ESCCH
	LEAX	CTABLE,PCR	Carga la dir. de CTABLE en X
	LDA	A,X	Toma el carácter de sustitución considerando como desplaz. el car. cont. en A
	BRA	FINISH	
ESCCH	INC	EFLAG,PCR	Activa EFLAG para avisar que se trata de un carácter Escape
FINISH	PULS	X	Restaura X
	JMP	OUTCH	Visualiza el car. cont. en A
	END		Nótese que al final de OUTCH, la instrucción RTS devolverá el control desde aquí al programa que llamó a esta rutina

Rutina de demora software

Subrutina para demorar (A) milisegundos

COUNT	EQU	195	Ver el cálculo
	ORG	$1000	
DEMORA	PSHS	B,CC	Guarda los otros dos registros afectados
LOOP1	LDB	#COUNT	Cuenta 1 ms
LOOP 2	DECB		Va decrementando...
	BNE	LOOP2	...hasta que B llega a cero
	DECA		Decrementa a A a cada ms...
	BNE	LOOP2	...hasta que A alcanza el cero
	PULS	PC,B,CC	Retorno

Decisión suprema

Los juegos de mayor aceptación suelen combinar elementos de juegos recreativos, de estrategia y de aventuras. "Psytron" es un buen ejemplo

Psytron es un juego complejo y absorbente que despliega unos excelentes gráficos de acción rápida, y cuyo dominio le llevará al jugador mucho tiempo. Éste asume el papel de Psytron, un dispositivo mitad hombre, mitad ordenador que dirige una colonia espacial en el planeta Betula 5. La colonia se compone de varios edificios, cada uno de los cuales posee una función específica. Los sistemas de apoyo a la vida son esenciales para los colonos humanos, y se requieren plantas energéticas para mantener el ordenador en funcionamiento. La instalación más importante es la planta energética principal, sin la cual toda la colonia quedaría paralizada. Mediante la utilización del teclado o bien de una palanca de mando, al jugador se le ofrece una "exploración" completa de 360º de la instalación. Cada edificio está dibujado cuidadosamente para conferirle mayor realismo a la panorámica.

Durante los primeros niveles de juego, *Psytron* se comporta de forma muy similar a cualquier otro juego recreativo. El usuario dispone de numerosas armas con las cuales repeler los ataques de invasores extraterrestres, y es en el cuarto nivel donde se hacen evidentes los elementos de estrategia del juego.

En el primer nivel, el Psytron controla un *droid*, que se emplea para destruir a los saboteadores de tres piernas que son "teletransportados" a la base en un intento de hacer volar las esclusas de aire que conectan entre sí los edificios de la colonia. El segundo y el tercer nivel de juego le ofrecen al jugador la oportunidad de derribar los platillos volantes de los extraterrestres; éstos se pueden coger uno por uno; no obstante, si se utiliza Disruptor, es posible eliminar de una sola vez todos los alienígenas que haya a la vista. Pero el Disruptor es algo inestable y hay un 10 % de probabilidades de que estalle cuando se haga uso de él.

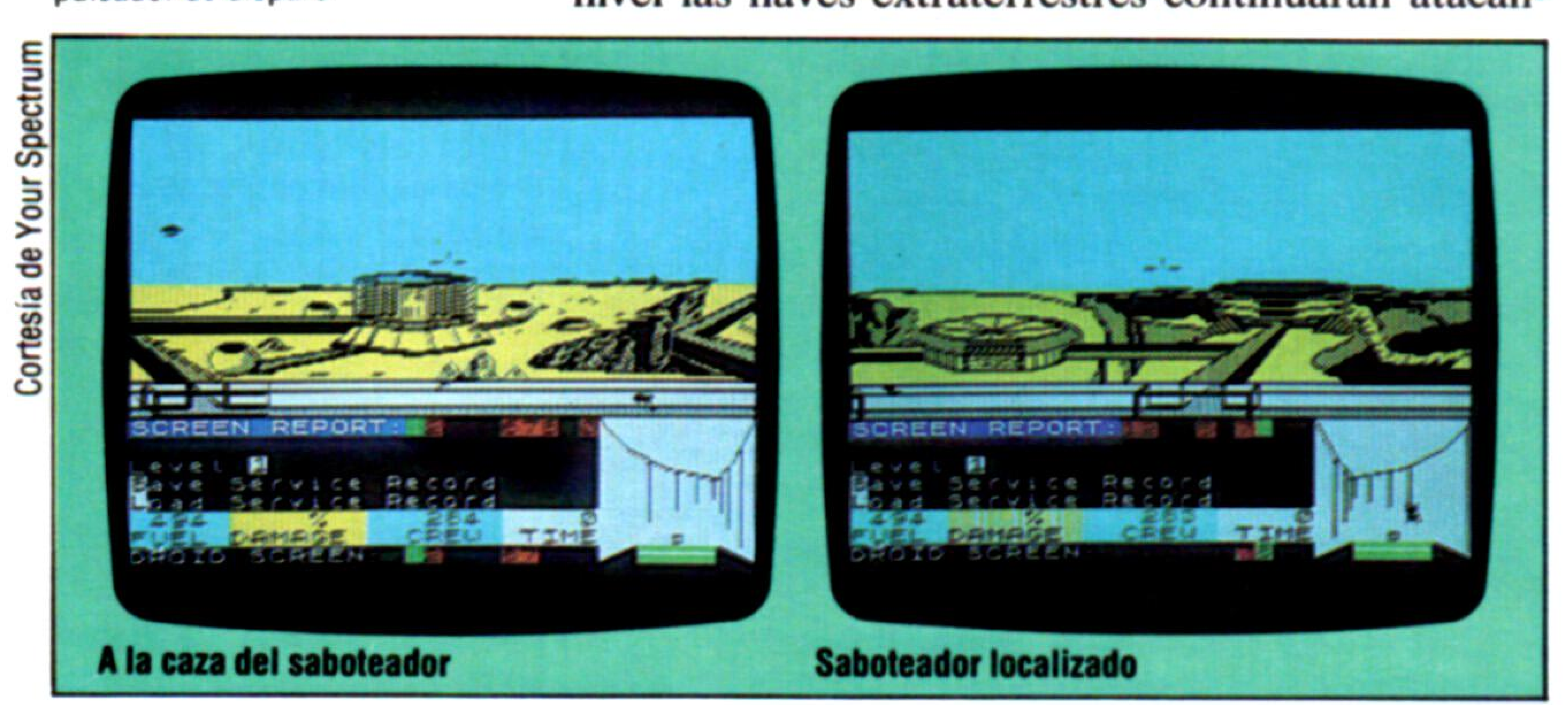
Una vez llegado el cuarto nivel, el jugador tiene la oportunidad de tomar decisiones estratégicas, basadas en la cuantía de los daños que hayan sufrido las instalaciones de la colonia. A lo largo de este nivel las naves extraterrestres continuarán atacando, bombardeando estratégicamente zonas vitales de la base y lanzando saboteadores en misiones suicidas. No obstante, en este nivel se introduce el "tiempo congelado". Mediante la simple pulsación de la tecla Return, el jugador puede "congelar" la acción para poder recibir y procesar informes acerca de los daños. Los factores a considerar incluyen el número de miembros de la colonia que han resultado muertos o heridos, el nivel de los suministros disponibles y los daños infligidos a la planta energética. Estando en tiempo congelado se pueden realizar reparaciones y se pueden destinar miembros de la colonia a aquellas zonas en las que sean más efectivos. En esta etapa es necesaria una cuidadosa administración de los recursos: Psytron debe tener en cuenta el hecho de que los grupos de reparación consumirán más alimento y más oxígeno que los colonos que no trabajen, y que podría ser necesario abandonar algunos edificios con el objeto de economizar combustible, aire y provisiones.

El quinto nivel le presenta al jugador la oportunidad de comunicarse con una nave de suministros, que podría hacerle llegar a la colonia provisiones vitales. Por consiguiente, se ha de tener especial cuidado en asegurarse de que los sistemas de acoplamiento no hayan resultado dañados en los ataques enemigos.

En el sexto y último nivel los factores estratégicos revisten máxima importancia. El único objetivo en esta etapa consiste en sobrevivir durante una hora, conservando la base intacta mediante la utilización de todas las facilidades que han sido introducidas en los niveles anteriores. Aumenta el número de naves atacantes, la acción se acelera y enseguida resulta evidente, a la luz de la sobrecogedora potencia de fuego de los atacantes, que es imposible mantener intactas todas las instalaciones de la colonia. Se deben tomar decisiones relativas a qué edificios se deben sacrificar; es de vital importancia, en particular, proteger la bahía de acoplamiento, dado que la misma permitirá el reabastecimiento de provisiones.

Los juegos por ordenador han recorrido un largo camino desde los días de *Space invaders*, y *Psytron* representa una alternativa exigente y absorbente a los juegos recreativos que hasta hace muy poco han venido dominando el mercado.

A través de los ojos del "droid" Estas escenas de *Psytron* muestran parte de la acción que se desarrolla en el primer nivel del juego. Mientras el *droid* persigue al saboteador a través de los corredores, la panorámica va girando a través de la base. Mediante la ventana situada en el rincón inferior derecho de la imagen, el jugador puede ver a través de los ojos del *droid*. Cuando el saboteador aparece en la pantalla, el jugador lo puede destruir accionando el pulsador de disparo



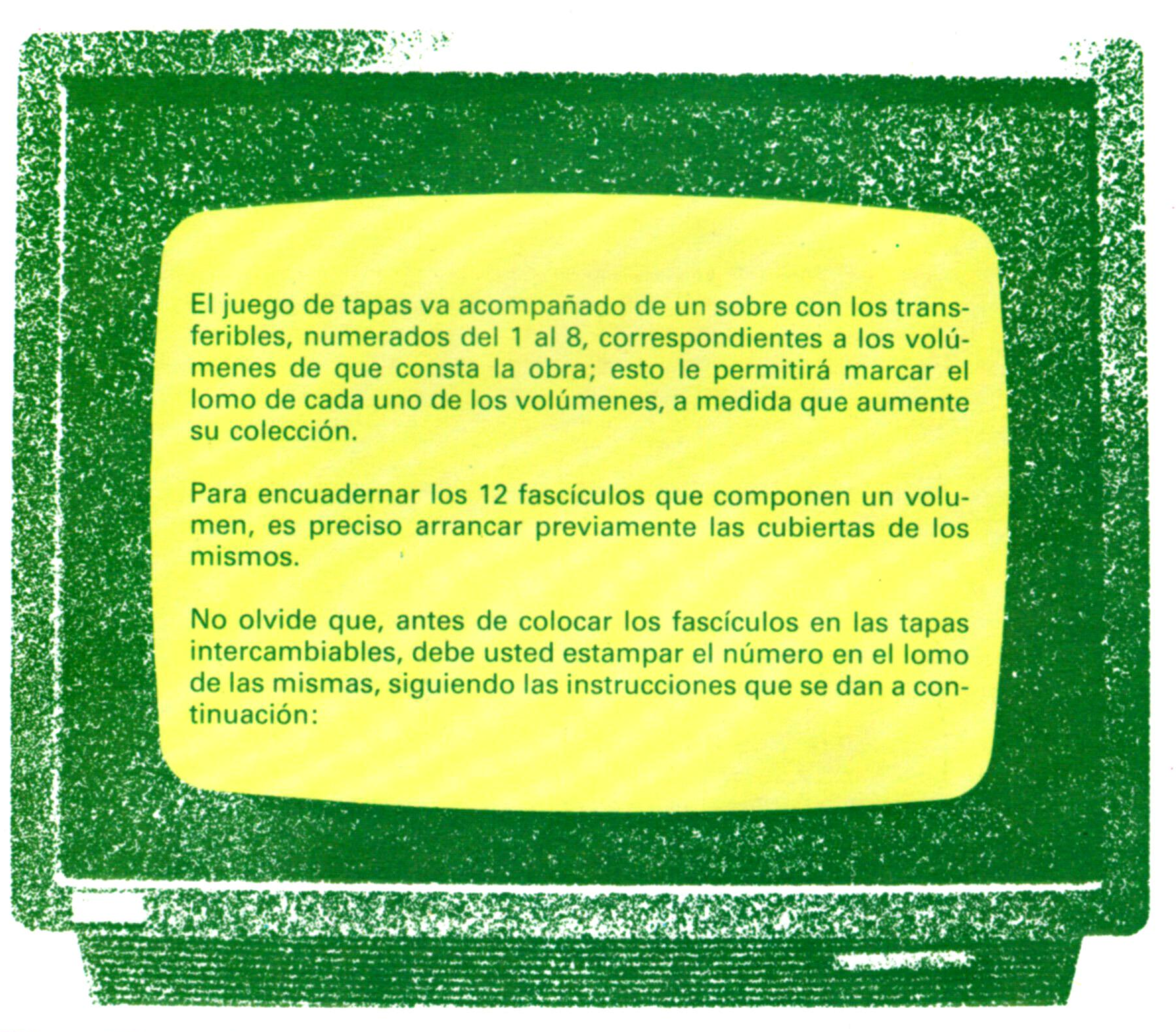
A la caza del saboteador — **Saboteador localizado**

Cortesía de Your Spectrum

Psytron: Para Spectrum de 48 K y Commodore 64
Editado por: Beyond Software, Competition House, Farndon Road, Market Harborough, Leics LE16 9NR, Gran Bretaña
Autores: Tayo Olowu y Paul Voysey
Palanca de mando: Opcional
Formato: Cassette (Spectrum), cassette o disco (Commodore 64)



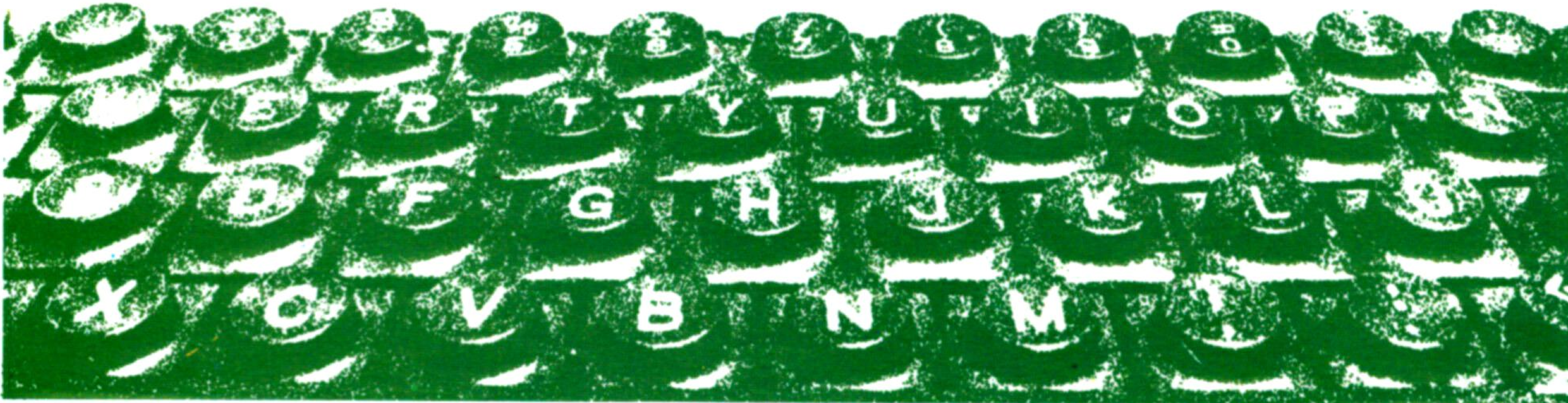
Con este fascículo se han puesto a la venta las tapas correspondientes al quinto volumen



El juego de tapas va acompañado de un sobre con los transferibles, numerados del 1 al 8, correspondientes a los volúmenes de que consta la obra; esto le permitirá marcar el lomo de cada uno de los volúmenes, a medida que aumente su colección.

Para encuadernar los 12 fascículos que componen un volumen, es preciso arrancar previamente las cubiertas de los mismos.

No olvide que, antes de colocar los fascículos en las tapas intercambiables, debe usted estampar el número en el lomo de las mismas, siguiendo las instrucciones que se dan a continuación:



1 Desprenda la hojita de protección y aplique el transferible en el lomo de la cubierta, haciendo coincidir los ángulos de referencia con los del recuadro del lomo.

2 Con un bolígrafo o un objeto de punta roma, repase varias veces el número, presionando como si quisiera borrarlo por completo.

3 Retire con cuidado y comprobará que el número ya está impreso en la cubierta. Cúbralo con la hojita de protección y repita la operación anterior con un objeto liso y redondeado, a fin de asegurar una perfecta y total adherencia.

Editorial Delta

Cada sobre de transferibles contiene una serie completa de números, del 1 al 8, para fijar a los lomos de los volúmenes. Ya que en cada volumen sólo aplicará el número correspondiente, puede utilizar los restantes para hacer una prueba preliminar.

10060
9 788485 822836